1 Experiences with a Smart Office Project

F. Bagci, F. Kluge, B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer
Institute of Computer Science, University of Augsburg
Eichleitnerstr. 30, 86159 Augsburg, Germany
{Bagci, Kluge, Satzger, Pietzowski, Trumler, Ungerer}@Informatik.Uni-Augsburg.DE

1.1 INTRODUCTION

A long time the technology in offices was limited to paper and ink since the first office environments. Paper is a robust, consistent and persistent information medium, that is easy to use and not fixed to a location. It is still used for communication internally inside the office building and externally with customers or other authorities. Also the location of offices was a strategic decision that considered traditional circumstances. You can find business areas in nearly each city supporting paper communication and face-to-face meetings. But location is expensive as is generally known. Compared with manufacture plants where work spaces are used nearly 100% of the day offices are occupied only 30%. In consulting companies where most of the employees work the most time outside the offices, the usage is even less than 10%. Reducing the amount of offices directly reduces the fixed costs.

Technology progress has fundamentally changed the work situation in offices. Information can be delivered over large distances easily within short time. What was earlier printed on paper became digitized and available online from almost everywhere. New devices were developed changing and supplementing paper work like printer, fax, and photocopier. Today an office without such devices can’t be imagined. Unfortunately this development did not decrease the number of offices holding the price of placement still very high.

The solution to this unsatisfying situation are flexible offices. In flexible office environments employees are not bound to a fixed location. The offices can be distributed dynamically among the actual demands. This is not an easy job to do if you want to guarantee a working place for every employee. But visions of ubiquitous and organic systems nurture the hope of automatically accomplishing the distribution and all its implicated management. More than this ubiquitous systems enable new services that can support office work in an unprecedented manner. Such
EXPERIENCES WITH A SMART OFFICE PROJECT

flexible offices need a sophisticated infrastructure to offer a seamless integration of the employees working environment at different locations. The flexible office concept we envision is described by the following scenarios.

**Flexible Office Scenario:** A sales representative needs a meeting room at his company for the next day. He places a reservation online specifying the time and number of attendees of the meeting. He will also need an office to work there for some hours.

The next day when the sales representative enters the building he takes his office equipment out of a store. The employee’s security badge is used to track his location within the office environment. He is guided to the assigned office by the Smart Doorplate System. If he passes a Smart Doorplate it shows an arrow to the direction of the assigned office. His container can also be localized by the location tracking system. When he reaches the assigned office he puts the container near one of the desks and the system reconfigures the environment (e.g. telephone, computer). The doorplate now displays his name to show that he is working in this office for that day.

Some minutes before the meeting the visitors arrive at the entrance. At the information desk the visitors receive a security badge and select the name of the person they want to visit. The visitors can also be tracked by the indoor location tracking system and are guided by the Smart Doorplates to the meeting room. Meanwhile the system informs the salesrep that the visitors just arrived and that they are on their way to the meeting room.

A flexible office organization, where office rooms are dynamically assigned to currently present employees requires a sophisticated software system that is highly dynamic, scalable, and context-aware. The considered computing environment within an office building consists of servers, PCs, laptops, PDAs, and sensor-nodes connected by wired or wireless networks. Because of the heterogeneity of the participating hardware and software systems, we chose a middleware approach, and because of the complexity of the system, we based our middleware on the autonomic and organic computing principles respectively, trying to overcome the high administration demands of such systems.

On top of the middleware are the services which can be grouped core services of the middleware and application services. In order to execute highly distributed and mobile applications we developed a ubiquitous mobile agent system. The paradigm of mobile agents offers a convenient approach to combine personal interests of users and the requirements of a ubiquitous distributed system. Mobile agents basically build the highest level of decentralization. Personal information can be encapsulated by a mobile agent and be used for location-based services on behalf of the user. The idea is that a user is accompanied by a virtual reflection in form of a mobile agent in the ubiquitous environment.

The next section describes the Smart Office Project in detail. It concentrates on the scenarios and hardware components attached in the offices. Sections 1.3 and 1.4 focus on the application level - the ubiquitous mobile agent system UbiMAS which serves as basis for some application scenario implementations. Section 1.4 introduces location tracking and location prediction. Section 1.5 introduces OCμ, the underlying organic computing middleware for ubiquitous environments. The

1.2 SMART OFFICE PROJECT

The long term target of our research is to provide a flexible office environment as platform for ubiquitous and organic computing applications. We chose our own office floor to build a smart facility. Each office has several devices like PCs, Laptops, PDAs, telephone, and sensor-boards. The high-light of the flexible office are the Smart Doorplates. We replaced the usual doorplates at each office door by electronic displays with touch-screens. Figure 1.1 shows a picture of one smart doorplate and figure 1.2 shows the floor of our institution with a smart doorplate at each door.

Fig. 1.1  Smart Doorplate

The Smart Doorplates can not only display information about the actual office but provide a convenient possibility to perform user-oriented and location-based services. Using the touch-screen the person can interact directly with the flexible office system. The following scenarios demonstrate applications of the Smart Doorplates (see figs. 1.3 to 1.5) in combination with a visitor/employee tracking system and additional sensors.

1.2.1 Smart Doorplate as signpost

The Smart Doorplates can act as signposts within the office building to direct an employee to his assigned office and to direct visitors to a sought employee, respectively. We assume that the visitor registers at an electronic reception as described in the introduction and that he selects the person he likes to meet. The visitor gets a security badge, and is directed to the employee’s office. The guiding system is implemented
by the Smart Doorplates. As soon as the visitor is in the vicinity of a doorplate, the doorplate points in the direction of the sought office. Assuming that a single visitor passes several Smart Doorplates on his way to the office, a direction pointer is sufficient and most appropriate. If the employee is not in his office but within the building, the Smart Doorplates direct the visitor (or colleague) to the current location of the employee.

Figure 1.3 shows the basic function of the Smart Doorplate presenting two office owners and some additional information. If an employee is in his office a small icon is shown in front of his name.

1.2.2 Visitor in front of the office and office owner present

In case of the presence of the office owner, several possibilities arise. The office owner is on the phone and may not be disturbed. The doorplate displays a phone sign at the employee’s icon to prevent disturbances. When the phone call is finished, the Smart Doorplate notifies the waiting visitor and ushers him in. If two or more colleagues share an office, it may happen that one colleague is busy, whereas the requested employee is ready to meet the visitor. The doorplate allows the visit if it is for the unoccupied office member using the information provided by the reception.

If the office owner is in a meeting within his room the Smart Doorplate displays that the office owner is present, but should not be disturbed. The office owner may receive a notice about the waiting visitor with the name of the visitor on his notebook or PDA, and he may answer by a notice, when the end of the meeting can be expected. All visitor messages are sent and received through the Smart Doorplate not disturbing the meeting.
If the office is used as a meeting room the doorplate may display an appropriate message that the room is occupied, which meeting it is, the list of attendance (see fig. 1.4), time when the meeting will end etc. Such information may be drawn from an electronic meeting protocol.
1.2.3 Visitor arrives in absence of office owner

If the office owner has locked his office and a date is shown in his electronic schedule, the doorplate may display his current location (in house, out of house or more details) and time when he is expected to be back. If the employee is located in a different office, the room number may be displayed and the surrounding doorplates can be used as direction pointers. Alternatively, the system could predict if the employee is coming back soon and recommend the visitor to wait (see fig. 1.5). The visitor may leave an oral or written message at the doorplate (microphone or touch panel presupposed). On return the office owner is notified about the message and may read it on the doorplate or within his office. Urgent messages may be forwarded to the current location of the office owner (e.g. by instant messaging over LAN or WLAN, or SMS over GSM).

![Fig. 1.5 Smart Doorplate in case of absence](image)

1.2.4 Smart Doorplate or foreign PC for e-mail retrieval

Usually an employee uses his office PC to retrieve e-mails. In this scenario the employee can arrange to be notified over the nearest Smart Doorplate if he expects an urgent e-mail. Because of security reasons he doesn’t want to receive all his incoming mails on a foreign doorplate or a PC in another office. He can use the doorplate or a foreign PC to set up a filter for incoming e-mails. The office system takes care about security concerns and informs the user when the expected e-mail arrives. The employee can than read the e-mail at the nearest located doorplate or on a foreign PC in an office. For authentication the user has to enter username and password. We have implemented a keypad (Figure 1.6) that provides a convenient approach to enter this data on the restricted touch-screen.
3.1.3 Das Dialogfenster

Das Dialogfenster wurde implementiert, um dem Benutzer kurze Nachrichten zuzuschicken. Es erscheint im Agentensystem, z.B. falls eines der Textfelder leer ist.

Abbildung 5: Das Dialogfenster

3.1.5 Smart Doorplate or foreign PC for file-transfer

In the same way like an e-mail the employee can choose a file that is placed on his office-PC and can transfer it to the actual Smart Doorplate or a foreign PC in an office. This application is very helpful if the user is for example in a meeting that takes place in a distant part of the building. Even if the user has no electronic devices to access his files he can transfer documents from his PC to the meeting room. He can make the document available for the persons in the meeting room, so each one can read or copy it.

1.2.5 Smart Doorplate or foreign PC for file-transfer

In the same way like an e-mail the employee can choose a file that is placed on his office-PC and can transfer it to the actual Smart Doorplate or a foreign PC in an office. This application is very helpful if the user is for example in a meeting that takes place in a distant part of the building. Even if the user has no electronic devices to access his files he can transfer documents from his PC to the meeting room. He can make the document available for the persons in the meeting room, so each one can read or copy it.

1.3 REFLECTIVE MOBILE AGENTS WITHIN SMART OFFICES

As seen in the scenarios above we propose that the smart environment takes care for storing and sending the personal information. The person is always accompanied by a mobile virtual object in the smart environment. So location-based services adapted to personal profiles can be offered. Certainly the ubiquitous system could be realized as a server-centric approach. But concerning personal movements and data this would lead to a big-brother-is-watching-you scenario where entities that gain access to the server would have access to all personal information. Regarding smart environments a central server could rapidly become a bottleneck because of the amount of clients and services running on the system. Moreover, a failure on the server would endanger the whole system.

The paradigm of mobile agents ideally fits into the decentralized approach. The mobile agent constitutes a virtual reflection of the user. Reflective agents accompany persons in the smart environment and carry user specific data. Employees have their
personal reflective agent, which resides in the environment and contains data about the employee. The data consists of basic user information like name, office room, etc. and security data like private and public keys, user names and passwords of the owner used for communication, data security, and access operations. Furthermore the agent stores context information belonging to the user and updates these data automatically.

The doorplates also serve as interface between user and agent. The user can instruct his reflective agent to perform services on behalf of himself. The reflective agent communicates with service agents and passes on the instructions. If the user moves to a new location the reflective agent migrates to the doorplate next to the user. The user location is determined by a combined tracking system that is described in section 1.4.

This mobile agent model is implemented in UbiMAS (Ubiquitous Mobile Agent System) [4, 3, 2]. Few other mobile agent systems exist for ubiquitous environments. Examples are the Hive system [21] that is a distributed agent platform for building applications by networking local system resources, and the Spatial Agents [35] that describes a framework where mobile agents follow their users as they move around and adhere to places as virtual post-its. The UbiMAS framework architecture is described in the following subsection.

1.3.1 UbiMAS Framework

The UbiMAS framework describes a skeleton of a mobile agent system based on a middleware. The agent system is consciously separated from the middleware layer in order to define a generic framework for a broad usage.

The services form the top layer of the middleware. A UbiMAS host is running as a service besides other services as e.g. a location tracking service, middleware services etc. The UbiMAS host represents a platform for mobile agents. It supports basic functions like starting, performing, and terminating agents and their communication.

We implemented an organic and ubiquitous middleware which uses the peer-to-peer system JXTA [33] as communication infrastructure. The middleware is described in detail in section 1.5.

UbiMAS hosts consist of two parts: the UbiMAS basic platform and the UbiMAS extensions. The UbiMAS basic platform defines abstract agent hosts and the interface for agent implementations. It realizes the basic communication functions between hosts and agents and several security concepts which cover host and agent security problems.

The UbiMAS extensions implement the application specific components, i.e. the reflective user agents and the service agents. Furthermore the communication functions are extended with secure agent-to-agent and agent-to-host messaging to fulfill the requirements of the Smart Doorplate application.

This separation in a basic and an extensions part eases the adaption of UbiMAS to a broad range of applications.
1.3.2 The UbiMAS Host Service

To host UbiMAS agents each peer node has to start at least one UbiMAS agent host as a service on top of the middleware (see Figure 1.7).

UbiMAS implements additional communication protocols for agent and host communication and for agent migration. All messages are acknowledged in UbiMAS which is not implemented by the base middleware.

If a middleware peer receives a UbiMAS type message it informs the host sending an event on which the host listens. UbiMAS hosts implement a message delivery engine which receives the events sent by the middleware and processes the incoming messages. If the host wants to send a message it forwards it to the message delivery engine where the appropriate header information is filled in, in order to avoid camouflage attacks.

If the receiver of a message is an agent the message delivery engine hands it on to the PoBox. The PoBox implements for each agent an interface called PoBoxAdder that allows the agents to send and receive messages. All communication between agents and hosts is handled over the PoBox using the PoBoxAdder. The PoBoxAdder describes the interface for putting messages into the queues. Besides the addMessage method of the PoBoxAdder there are no method references between hosts and agents.

The PoBox has a queue for incoming messages for each agent. In the same manner each agent has a queue for messages sent by the PoBox. The queue lengths are managed dynamically by the possessing entity. This approach offers various security features which are described in more detail in [2].

Fig. 1.7 The UbiMAS Host Architecture
EXPERIENCES WITH A SMART OFFICE PROJECT

The agent hosts can form alliances in form of peer groups using the services of the middleware group manager. If two peers want to build a peer group they have to arrange new pipes which are built using a pipe advertisement. The advertisement contains a unique pipe ID. Only the peers which know the pipe ID can receive the messages sent over the pipe. Each agent host which wants to join a peer group must send a request. The peer group members can decide if a request will be accepted or rejected. The agent hosts within a peer group communicate over a secure communication protocol.

Different peer groups can build a partnership. Applied to an office building each floor could build a peer group, and floors of the same institution could form an alliance. This makes it possible for agent hosts to communicate with hosts from other peer groups. Messages to hosts outside the peer group are not secured. Beside UbiMAS there are other services, e.g. Location Services, using the same middleware for communication. Mobile Agents can register for events of these services and will be informed when new events occur. In this way agents can be notified by sensor events from the environment or location events of particular persons.

1.3.3 UbiMAS Mobile Agents

Agents in UbiMAS are started as single threads on the actual host. An agent reacts to messages, i.e. the agent is in a loop where it waits for incoming messages. A UbiMAS host can manage several agents. There are two types of agents in UbiMAS: user-agents and service-agents. Both agent types are derived from an abstract agent that defines different basic methods for communication and security functions.

Agents can communicate with each other using messages. An agent must implement functions which are performed when a specific message is received. If an agent doesn’t know the message type it ignores the message. The abstract agent defines methods for creating and sending messages. The agent hosts serve here as mediators. Messages addressed to a locally available mobile agent are forwarded directly to the recipient by the local PoBox. If the agent is on another host the message is transferred to this host.

Each agent is identified by a unique ID. UbiMAS supports message encryption. Each host owns a certificate, a public, and a private key. Each agent has an own key pair for message encryption. The abstract agent defines further data structures for security keys of the agent and the public key of the actual host. Furthermore there are methods for requesting public keys of other agents or hosts.

The security architecture of UbiMAS aims to protect both the agents and hosts against malicious behavior. A secure system is essential for the acceptance of the reflective mobile agent approach because personal data is sensitive. Evaluations showed that in a smart office environment with 10 MBit/s Ethernet connections agent transfer with/without security features is adequate [4].
1.4 LOCATION TRACKING AND PREDICTION

1.4.1 Location Tracking

For complete operation, the Smart Office system needs a service that provides the current positions of the users. This service should at least be able to determine the room a user is currently staying in. Therefore several location tracking techniques exist. We will give an overview of some alternatives before explaining our own approach and experiences.

Pathbreaking work on positioning systems was done by Want et al. [46] with their Active Badges. They employed a room-accurate positioning system based on IR LEDs and sensors. A radio technique, wireless LAN, was employed in the RADAR system by Bahl et al. [5]. Especially this work discusses a wide range of methods for analyzing the measurements. Other radio techniques were analyzed by Feldmann et al. (Bluetooth, [10]) or Ni et al. (RFID, [22]), among others. For all mentioned techniques a medium error of about 2-5 m was reported. ARIADNE [18] promises to amend this, and also not to need large overhead for the recording of a reference map through the automatic construction of radio propagation maps. Here, the medium error reported drops to about 1 m.

Definitely better outcomes can be gained by the use of ultrasonic [13, 39]. Here, the error lies at 10 cm resp. 50 cm in 95% resp. 90% of all cases. However, these gains must be seen alongside with the high costs for the technical equipment and its installation.

Due to the high inaccuracy or the high costs for existing systems, we decided to build our own location tracking system. Therefore we used the Embedded Sensor Board ESB 430 [38], with a TR1001 radio transceiver.

The test bed consisted of three rooms of about 80 m², where we placed three sensor boards as infrastructure. A fourth board was worn by the mobile user. We used a passive infrastructure that acts as a receiver for the signals sent by the mobile board. Thus, the chronological assignment of the signals onto the positions was made easier. With this configuration, we recorded a reference model of over 30,000 data sets. In the rooms, we had 70 measurement points, and at each point we collected data for four orientations. By recording several data sets at each point, we hoped to outweigh the jitter of the radio transceivers at least a little.

For the calculation of positions, we compared three methods: the Nearest Neighbour in Signal-strength Space (NNSS) search that already was used in RADAR [5], a statistical analysis based on the density of a normal distribution (STAT), and a randomized analysis, where a point is chosen randomly for each measurement. The last one was used as a reference method to see how good we can guess the position.

An evaluation using the three methods showed the following [20]: As expected, randomized analysis performed worst. NNSS and STAT nearly performed equal with a median deviation of about 2.5 m, whereas NNSS performed slightly better. In further calculations we analyzed the influence of the number of neighbouring points on our results. As has shown up, especially when dealing with low numbers of neighbours improvements can be received with each additional neighbour. Coming
EXPERIENCES WITH A SMART OFFICE PROJECT

to higher numbers, the improvement is negligible. Furthermore we analyzed the influence of the size of the reference database (number of reference points) and the number of measurements at each point. As has shown up, both only minimally influence the position calculated.

Further analysis of the position calculation showed that only above 60% of all calculations met the correct room. So if it was possible to estimate first the room with a 100% accuracy, the position calculation could be improved. To confirm this assumption, we used the available IR sender and receiver on the boards to determine the room. Evaluations showed, that the median deviation decreased from 2.5 m to below 2 m. The accuracy of radio-based systems is very sensible to changes in the environment, mobile walls, opened doors, and even air moisture. Thus, a combination of two methods by fusion of both sensor systems that are for themselves not that accurate, may increase overall accuracy.

For real use in the Smart Office project, this method is still too inaccurate. A great problem is the jitter of the received radio signal strength. Here, the use of a better transceiver might bring some improvement.

1.4.2 Location Prediction

Can the movement of people working in an office building be predicted based on room sequences of previous movements? In our opinion people follow some habits, but interrupt their habits irregularly, and sometimes change their habits. Moreover, moving to another office fundamentally changes habits too. Thus location prediction methods need to exhibit some features: high prediction accuracy, a short training time, retention of prediction in case of irregular habitual interrupts, but an appropriate change of prediction in case of habitual changes.

For our Smart Office application we used benchmarks with movement data of four persons over several months. The benchmarks are called Augsburg Indoor Location Tracking Benchmarks. They are publicly available [23], and are applied to evaluate several prediction techniques and to compare the efficiency of these techniques with exactly the same evaluation set-up and data.

Our aim is to investigate how far machine learning techniques can dynamically predict room sequences and time of room entry independent of additional knowledge. Of course the information could be combined with contextual knowledge as e.g. the office time table or personal schedule of a person, however, we focus on dynamic techniques without contextual knowledge.

Several prediction techniques are proposed in literature — namely Bayesian networks [17], Markov models [6] or Hidden Markov models [34], various neural network approaches [12], and the state predictor methods. The challenge was to transfer these algorithms to work with context information. In our research we investigated five approaches, a dynamic Bayesian network [29], a multi-layer perceptron [45], an Elman net, a Markov predictor, and a state predictor [25]. In the case of the Markov predictor and the state predictor we additionally used a version which is optimized by confidence estimation [26] and enhanced by various hybrid predictor models [28]. A comparison of these methods can be found in [24, 27].
Time of arrival at the predicted location depends on the sojourn time at the current location plus the rather constant time to move to the predicted location. The sojourn time can be modelled into a Bayesian network \[29\]. We tested also a time prediction which calculated the mean and the median of the previous sojourn times within a location. The best results were reached by the median. The time prediction is independent of the location prediction method and can easily be combined with any of the regarded methods.

1.5 ORGANIC COMPUTING MIDDLEWARE FOR UBIQUITOUS ENVIRONMENTS

OC\(\mu\)\(^1\) \[41\] is designed with the goal in mind to facilitate the device independent application of organic computing demands in ubiquitous environments where we expect a heterogeneous collection of devices with diverse capabilities of computing power, memory space, and energy supply.

Beside the design of the middleware we investigate self-configuration, self-optimization, self-healing, and self-protection within the middleware. The self-x properties which are implemented as services can be used as needed. The overall architecture of OC\(\mu\) is shown in figure 1.8. The architecture of the middleware as well as the self-x properties are described in more detail in the follow sections.

---

\(^{1}\)OC\(\mu\) is an acronym for Organic Computing Middleware for Ubiquitous Environments
1.5.1 OCµ Middleware Components

The basic architecture of OCµ is comparable to other state of the art middleware systems. It is comprised of three layers. The lower level (*TransportConnector*) is responsible for the delivery of the messages to other nodes on different communication infrastructures. The middle layer (*EventDispatcher*) is capable of finding the accurate recipient of a message that was sent by another service either locally or from another node. The applications as well as some basic services of the middleware like the configuration service and the discovery service reside on the top level.

There are three additional parts in the middleware that differentiate OCµ from other middleware systems: first a sophisticated monitoring on both lower levels of the middleware, second the Organic Manager with the self-x services, and third a typed messaging that adds further freedom in terms of message delivery and service requests.

To use the offered capabilities of OCµ the applications should be separated in services which can be distributed on the nodes of the network. Future ubiquitous systems will be comprised of many computing nodes so applications should be composed of services instead of being a monolithic block of software. The advantage is twofold. The services of an application can be reused much easier than parts of a code and the services can be distributed over the network to increase the computational performance of the application.

For the Smart Office scenario the UbiMAS agent host, location tracking, and location prediction are some of the implemented application services.

**TransportConnector:** To decouple the middleware from the underlying communication infrastructure the TransportConnector employees specific TransportConnectors which must be implemented for the different communication infrastructures. In the current implementation we use JXTA [33] and provide a JXTATransportConnector. The peer to peer approach fits best the transport functionality used by OCµ.

The implementation of a TransportConnector can be replaced depending on the given communication infrastructure which is transparent to the rest of OCµ and the applications built on top of it. It is also possible to use multiple TransportConnector implementations for different communication infrastructures at the same time (i.e. CAN-Bus, Serial Line).

**Event Dispatcher:** The Event Dispatcher is responsible for the message delivery between the services. The Event Dispatcher offers to services the functionality to send messages and to register themselves as listeners for specified types of messages. A service can register for different types of messages and will be informed in case of an incoming message with one of the registered types.

The Event Dispatcher handles the delivery of broadcast and unicast messages. It knows whether a message can be delivered locally or if the message must be sent to a remote node. This information is collected during runtime explicitly by querying a special service if its location is unknown or implicitly by collecting the information from service advertisements which are exchanged between the nodes if a service is started. Each service has to register at the Event Dispatcher and must provide a
service advertisement that holds some basic information describing the service. The Event Dispatcher propagates the service advertisement to the other nodes.

**Services**: A service needs to implement a special interface to participate in OCµ and to receive messages delivered by the Event Dispatcher. The interface also offers the functionality to send messages.

We differentiate between two kinds of interfaces for services. The simple service which has the full functionality needed to communicate within the system and a relocatable service. Beyond the functionality of a simple service a relocatable service can be transferred to another node whereas a normal service is bound to the node it was started on. The binding on a special node is important for services that need a special hardware or software environment (e.g. fixed sensors or databases).

**Service Proxy**: To forward messages for a service that was recently relocated to another node the Service Proxy is used. It has a limited lifetime and automatically dies after that time. The lifetime depends on the validity duration of a service advertisement which was created when the service was started on this node. This mechanism prevents the system from sending a surge of messages each time a service was moved to another node. We try to protract this overhead as long as possible. It is not necessary to update the information about a service on all nodes immediately after a service has moved. If another service wants to use the moved service it sends the request to the Service Proxy which forwards the message to the new location. The answer is sent back directly to the requesting node which can now update its entry for this service.

A node, whose information about a moved service exceeds the expiration time of the service advertisement, might be able to reach the Service Proxy which has the same lifetime as the service advertisement. If the Service Proxy can not be reached the node must discover the service anyway. So the overhead of updating all nodes can be shifted to the point where it can’t be avoided anymore, but it is always less then a broadcast to all nodes.

**Organic Manager**: The OCµ architecture is a observer/controller architecture where the Organic Manager including the self-x services takes the part of the controller and the monitors are the observers. All relevant information collected by the local monitors about the services and the node itself is stored at the Organic Manager in the Monitor- and SystemMonitor Information Pool, respectively. The Information Pools provide the collected information to other services and the self-x services employing a publisher/subscriber mechanisms. The Information Pool informs the subscriber if new information arises someone has subscribed for.

### 1.5.2 Monitoring

To get information about the services, resources and the node itself, monitoring is a vital point in OCµ. To avoid the communication overhead of a centralized monitoring all monitors collect information locally on every node.

To obtain as much information as possible it is necessary to use more than just one monitoring point and to use specialized monitors for different tasks. OCµ not
only uses a single monitor on every level (System, Transport Interface, and Event Dispatcher) but also uses Monitor Queues to add and remove monitors as needed.

For a fine-grained monitoring OCμ offers the possibility to add monitors for incoming and outgoing messages at the Transport Interface and the Event Dispatcher. Different Monitor Queues exist for each direction. The advantage of the Monitor Queues is that monitors can be added and removed as needed and that the monitors can be kept simple and fast. Furthermore the Monitor Queues offer the possibility to add monitors in defined orders to build processing queues for the messages.

**Monitoring on the Transport Layer:** The monitors at the Transport Layer are responsible for monitoring transport dependent information. Incoming and outgoing messages for example can be used to monitor the latency between peers and the amount of data exchanged between services. This information can be used by metrics to decide if it is better to run a service locally or at a remote node.

**Monitoring at the Event Dispatcher:** As the Event Dispatcher delivers incoming messages and outgoing messages it is the perfect instance to monitor service dependent information concerning the message exchange.

If a service requests information from another service it can measure the local and remote response time. It can also collect alive information about the services by monitoring the messages to assure that a service is still available. This prevents the overhead of alive messages by just looking at the ongoing communication.

**The System Monitor:** The System Monitor is used to gather information about the computing platform a node is running on. The information about i.e. memory, processing power, and communication capabilities might be vital for the metrics to reason about the distribution of the services.

Currently there exists no interface for Java to collect system dependent information. Various management APIs exist such as SNMP, JMX etc. to collect information about a system. To monitor the information about the local node we use the parts of the JMX and some native code to provide the requested information.

### 1.5.3 Typed Messaging

The communication of systems like CORBA or JINI is based on method invocation. They use the stub/skeleton principle to guarantee a consistent view on the methods of a remote object. The disadvantage of this approach is that the stub must be known at development (compile) time of a service or application and that a new service must exactly implement that interface.

The communication in OCμ is based on the exchange of messages and not on method calls. An EventMessage consists of MessageElements where each MessageElement has a name and a value it encapsulates.

In OCμ the parameter types must be defined like in stub/skeleton systems, but the order in which they appear is not relevant. Another advantage is the extensibility of the communication. A service can handle requests with a limited count of parameters as well as a request containing an extended set of parameters. This means that an extended version of a service, which can handle more parameters than the previous version, can be smoothly integrated into the system without notice to the rest.
If an interface is changed the stub/skeleton systems require a recompilation of all affected components. If an object calls a remote method, stub/skeleton systems identify the desired method through the introspection mechanism which tries to find a method with the correct name, parameter count as well as the order and types of parameters. This is not the case for OC\(\mu\) because the services register for the types of the messages they are interested in and they provide the capability to handle messages with different sets of parameters.

OC\(\mu\) supports two methods for the message delivery. A service can be directly addressed or a message can be typed, resulting in a delivery to all services that have registered for that type of message. If a message is directly addressed to one service the Event Dispatcher delivers the message only to that service leaving out all other services that might have registered for the type of that message. If a message is sent as a typed message the Event Dispatcher selects all services that registered for that message type and sends the message to all of them. The message types can be user-defined and hierarchical. Another difference between OC\(\mu\) and the stubs/skeleton version is the nonblocking nature of requests. All messages are sent asynchronously, thus if a service sends a request to another service it is not blocked.

1.5.4 Self-Configuration

The self-configuration is used to find a distribution of the services of an application such that the Quality of Service (QoS) is as high as possible. A configuration description contains all the services and their resource requirements. This configuration description is flooded into the network and every node calculates the QoS for every service locally and orders them in descending order. Those services which can not be provided due to resource constraints are left out and collected separately.

The configuration description can have so called constraints. The constraints are requirements for the application that can be described in form of a mathematical for-all quantifier i.e. “Every node with a Smart Doorplate should start the smart doorplate service”. These requirements are treated prior to the normal service assignments.

After the quality of the services is calculated one node starts to send an assignment message containing the id of the service, the id of the node, and the QoS the node will provide for the service. This message is broadcasted to all other nodes. The receivers of this message mark the service as assigned in their list. Then another node might send an assignment message. This carries on until all services are assigned to the nodes.

During the assignment of the services, one node might provide a service with a higher QoS, thus it sends an additional assignment message which overrides the original assignment in the same manner as a normal service assignment. If a conflict occurs because two nodes want to assign the same service a conflict resolution mechanism is employed to solve the conflict without any additional messages. The assignment messages have four extra values which are used in case of a conflict. With this information every node can decide locally which one of the conflicting nodes should get the service. The conflict resolution mechanism is a five stage procedure...
where the next stage is used to find a solution if the values of the former stage were equal for both nodes.

After all services are assigned a verification step is used to assure that all nodes have the same assignments of the services. If a node receives a verification message it compares the contained information with the local assignments. If the QoS of the local assignments is better a further verification message is sent to inform the other nodes about the updated information. If the local assignments are worse, the improvements of the received configuration are incorporated. After the verification step every node can start the services it has assigned.

Evaluation results [19, 42] based on simulations and on running the self-configuration within the OC$\mu$ middleware showed, that an average amount of 1.5 messages per service suffices to assign all services to the nodes of the network.

The algorithm is capable of finding all unsatisfiable configurations. All other satisfiable configurations were successfully assigned to the network during the simulations as well as the experiments in a real OC$\mu$ setup. The QoS of the overall assignment is not optimal, but at least good enough to start the application. The effort needed to assign all services depends only linearly on the amount of service given in the configuration.

1.5.5 Self-Optimization

After the initial configuration of the system the services and the nodes are monitored by the corresponding monitors. The collected information is used to further optimize the system in terms of predefined resource parameters. The parameters used for the self-optimization can differ from those used for the self-configuration. During runtime parameters like CPU and memory usage, and network bandwidth are of interest. The self-optimization tries to relocate the services in a way that these resources are consumed equally by all nodes of the system.

The idea of the self-optimization is based on the human hormone system. Hormones (information) are produced in the cells (nodes) and released to the blood circuit (communication between the nodes). The hormones bind to cells with a matching receptor (monitor) and may trigger an action inside the cell (relocation of a service).

The digital counterpart of the human hormone system are the digital hormone values which are piggy-backed on the messages leaving the nodes. The digital hormone values are produced by a metrics which takes the previously mentioned parameters into account. The monitors of the receiving nodes extract the information carried by the messages.

Depending on the received load information of the remote node the self-optimization of the local node decides whether a service should be relocated to the other node or not.

Simulation results of networks with 1000 nodes showed that the self-optimization scales linearly with the amount of nodes. The self-optimization reaches 98.4% of the theoretical optimum with a minimum of service relocations. Furthermore the self-configuration is aware of the dynamic behavior of the services and can adopt to changes in the resource consumptions.
Although self-optimization employs a rather simple approach where information is processed only locally and no central component is used to control the self-optimization, it produces excellent results and scales linearly with the size of the network. The last point might be the most important one concerning ubiquitous environments where we expect a huge amount of inter-connected devices.

We developed and simulated four different Transfer Strategies [43]. The best one reaches about 98.4% of the theoretical optimum concerning the mean average error. We can show that the amount of transferred services of this is nearly the amount of transfers needed to reach the optimum. So if all services would be of the same size concerning the resource consumption we would reach the optimum with a minimal amount of service transfers. The amount of service transfers can be further reduced by an adaptive barrier which suppresses those relocations that do not add a significant gain to the overall optimization.

We also simulated a dynamic service behavior i.e. we considered a change in the resource consumption of the services. First we simulated a periodic behavior of the services where services change their resource consumption for a given percentage of a period and return to the original value for the rest of the period. Second we assumed a constant increase in the resource consumption during a predefined time. After that time the services rest at the new load level for a while and return to the normal level also within the same predefined time. It turned out that the Hybrid Transfer Strategy, which uses the dynamic barrier and an load estimator to calculate the average load of the network, can perfectly adapt to the dynamic behavior of the services and thus suppresses unintended service relocations. Experiments in the OC\(\mu\) middleware [41] showed even better results than the simulation did.

1.5.6 Self-healing

The task of self-healing is to assure that the Smart Doorplate system meets some defined conditions as far as possible. In our case we have to guarantee that all services as determined in the configuration of the system stay available.

Self-healing can be divided into (1) the detection of unwanted conditions and (2) the recovery from these conditions. We identified the following tasks of a self-healing mechanism:

**Failure detection:** Failure detection provides information on failures of components of distributed systems. A failure detector enables an OC\(\mu\) node to detect the crash of a node within a Smart Doorplate environment.

**Grouping:** The responsibilities for failure detection and other monitoring tasks have to be arranged intelligently in a way that assures secure monitoring with low overheads. This organization of the failure detection services is called grouping and states which OC\(\mu\) nodes monitor which other nodes in the system.

**Automated Planning:** Failure detection and Grouping refer to the detection of unwanted conditions. Automated Planning and scheduling can be used as an intelligent mechanism to recover from these conditions. The planner uses the
current situation of the Smart Doorplate system as input and outputs a solution how to recover to a state of the system that reestablishes the full functionality or abides the main functions of the system as long as possible.

**Scheduling:** A scheduling component is working hand in hand with the automated planner. It assigns the steps of the generated plan to OC\(\mu\) nodes and controls the recovery process.

**Distributed datastore:** We chose a datastore service to save data in a distributed manner within the OC\(\mu\) middleware and refrained from an automatic backup service which might be too resource-consuming for ubiquitous environments. Services can use this distributed datastore to save their context persistently i.e. services have to decide themselves when and which information should be stored. Thus a crashed service has the possibility to recover its last state.

Failure detection and distributed datastore are shortly described. Grouping, automated planning, and scheduling are work in progress.

**1.5.6.1 Failure detection:** Failure detectors provide information on failures of components of distributed systems. A failure detector enables an OC\(\mu\) node to detect the crash of another node which represents a tough challenge in distributed systems like our Smart Doorplate environment.

So far we developed and simulated a new adaptive accrual failure detection algorithm [36] and also investigated some variations of this algorithm [37].

Failure detectors in distributed systems require a mutual monitoring by heartbeat messages respectively by application messages from the monitored node within a certain time interval. Adaptive failure detectors [8, 7] are able to adjust to changing network conditions. The behavior of a network can be significantly different during high traffic times as during low traffic times regarding the probability of message loss, the expected delay for message arrivals, and the variance of this delay. Thus adaptive failure detectors are highly desirable.

The principle of an accrual failure detector [14] is not to output whether a node is suspected to have crashed or not like common failure detectors. Rather they give a suspicion information on a continuous scale whereas higher values indicate a higher probability that the monitored node has failed. Thus accrual failure detectors decouple monitoring and interpretation. That makes them applicable to a wider area of scenarios and more adequate to build generic tools.

The algorithm we developed is based on a contract defining the monitored node sending heartbeat-messages to the monitoring node every timestep. The monitoring node analyzes the inter-arrival times of the heartbeat-messages using statistical methods. As a result it outputs the probability that the monitored node has failed based on its previous behavior.

Our algorithm is characterized by very low computational demands on the processing power. This allows an application in a broad variety of hardware settings. It also shows very good results in comparison to other failure detectors, especially in the case of message loss. Comparing our algorithm to other state of the art failure
detection algorithms [8, 7, 14] shows that it significantly outperforms all other algorithms in certain important settings [36, 37]. Furthermore, in contrast to common failure detectors, it outputs a probability that a node has crashed, not only a boolean value. This is an ideal basis to build very flexible failure detection services in a variety of environments.

1.5.6.2 Distributed datastore: To ensure that a crashed and recovered service can access the last saved state, the OCμ middleware offers a storage service which itself exposes self-healing features to overcome node failures.

A distributed datastore spreads the data of the services on different nodes to add redundancy to the stored data. In case of a failure a service asks the distributed datastore about the data and the datastore is responsible to get the latest version of the service’s data.

The self-healing of the data storage itself is based on the measurement of some resources like memory consumption and communication bandwidth. The resources are measured during runtime and the data of the services is distributed considering the actual resource consumptions. But not only the resources are of interest. The self-healing also takes the average failure rate and the measured online periods of the nodes into account to raise the dependability of the data storage. The average failure rate is applied to rate a node’s trustworthiness. The lower the failure rate of a node, the higher the node is rated in terms of trustworthiness. The online time is especially interesting in environments where nodes are not online 24/7 like e.g. the PC’s of employees in an office environment. The online time is measured during the day and the start of the next offline period is predicted. The remaining online time is used as a further parameter to rate the node. The longer the time until the expected offline period, the higher is the node’s rate.

We simulated different implementations of the distributed data store [9]. The target of the simulation was to show how many failures are produced within different environmental settings. A failure is either a read operation from the data store which results in none or outdated data. Simulations with 100 nodes showed that the failure can be reduced to 0 even if we assume that one node of the network crashes every 36 seconds. A failure of 0.02% is achieved if the failure rate is raised such that a random node of the network fails every 18 seconds. Assuming an average failure rate of 18 seconds means that every node fails at least after half an hour. The interesting point about the distributed data store is the fact that only two additional nodes must hold a copy of the data to achieve the mentioned results.

1.5.7 Self-Protection

Research on self-protection in OCμ is so far restricted to the detection of new and therefore potentially malicious messages by techniques from computer immunology [11]. Current violation detection systems are normally rule based or use signatures which rather result in a static detection of intrusions. Computer immunology opens up new ways and methods to recognize new intrusions like our biological immune system does. In the area of self-protection we are going to design a protection architecture
EXPERIENCES WITH A SMART OFFICE PROJECT

that enables all members in the middleware to identify intrusions without the need of central instances. Once an intrusion is detected by one peer member, the nodes will be able to eliminate threats or exclude malicious services from the middleware on their own by effective communication strategies. As for now we investigated the usage of artificial antibodies to detect messages from intentionally or unintentionally malicious or yet unknown services. The biologically inspired technique of computer immunology extracts ideas from our human immune system to develop an artificial counterpart [1].

A look at the functionality of the biological immune system shows that the basic requirement of such a system is to distinguish between harmless objects called \textit{selfs} and harmful objects called \textit{nonselfs} [15]. Matching works due to different protein patterns on the surface of the unknown objects and the antibodies. All mammals have a \textit{thymus} which is aware of all selfs known in the body. Antibodies are continuously created with random protein surfaces and get singled out with \textit{negative selection}. If such a cell is activated in the thymus it will be destroyed [15] otherwise it will be released to the body to protect it against a specific type of intruder. The opposite process called \textit{positive selection} is also used in the body [40] which focuses a T-cell receptor to recognize peptides bound to molecules of the major histocompatibility complex (MHC). It works similar to the negative selection process but in difference it singles out all protein surfaces which \textit{can} bind to selfs in the thymus. Those receptors which do not bind are destroyed. The biological immune system is extremely distributed besides the centralized work of the thymus.

Our middleware services solely communicate over messages and because of the sandbox mechanism in the Java environment, which is the implementation basis of OCµ, those messages are the only way to impact the system right now. Due to that fact our middleware-based immune system doesn’t aim on proteins but on binary messages. The basic requirement of the artificial immune system is also to distinguish between foreign nonself messages and messages which are tolerated in the system because they belong to self. Instead of storing the huge amount of information about all selfs on every node we want to filter out messages by comparison to short bit strings. Thus we first developed antibodies analogous to the r-chunk method [40]. The antibodies are represented by a short bit string of length \(r\) which embodies the receptor. Additionally they have a specific offset \(o\) where they start their comparison to the messages. Before the antibodies get released to the system they are perambulated with the negative selection process. Instead of waiting for all newly created antibodies and probodies until they perambulated the thymus like in the human body we preferred to generate them in a structured way. This works because the middleware OCµ is aware of all its self messages and thus it can create all receptor patterns not used by any self message at a specific offset. We also do not compare the whole message because this would lead to an infinite amount of selfs due to the changing data in the message. Therefore we only consider the header of the messages which consist of sender and message type.

First off we investigated the relation between the design of the antibodies (e.g. different receptor lengths and offsets) and the resulting detection rate. We designed a simulator to measure the effectiveness and reached detection rates of up to 99.6% of
artificial and randomly constructed intrusive messages. Best recognition rates were reached when receptors with a length of $r \simeq \log_2(n)$ are used in a system which contains $n$ self messages [32]. We also recognized that the storage of receptors needed a lot of space and also comparison takes a long time. Therefore we found optimizations of our technique in both areas. For minimizing the space complexity we combine similar receptors. This results in receptors with wildcards and in our test runs we were able to eliminate 30% of the space needed for storing the antibodies. This also helps minimizing network usage when exchanging antibodies between nodes. To enhance the comparison itself we organize the receptors of specific offsets in a tree structure. This results in a constant time complexity for comparison and in simulation runs it gained a speedup of 30 [30].

The opponent of the antibodies is the group of so-called probodies. They are designed the same way as the antibodies but in difference are generated with the use of the positive selection mechanism. We found out that a combined usage of antibodies and probodies reaches better results than only using antibodies but also has some drawbacks. Because of the nature of the selection mechanisms all possible probodies at a specific offset always have to appear completely [31].

1.6 CONCLUSION

Flexible offices can drastically reduce costs by dynamically assigning office rooms to present employees. This organization requires a sophisticated software system that is highly dynamic, scalable, context-aware, self-configuring, self-optimizing, self-healing, and self-protecting. In this paper we presented our research experiences with a flexible office environment which satisfy these requirements. Due to the organization of our department with fixed offices we cannot investigate the flexible office paradigm itself. However, we developed and implemented a suitable ubiquitous middleware, a mobile agent system for encapsulation of user context, a location tracking system, and location prediction techniques. Office rooms can be dynamically assigned to currently present employees. Applications are realized through mobile agents which offer a possibility to encapsulate personal information of a user and to perform location-based services of the ubiquitous system in the name of the user. Persons and objects are tracked by a location tracking system that is integrated as an additional ubiquitous service. Moreover, a location prediction service provides assumptions where an absent office owner will be next and when he/she will be back. To increase manageability of the complex system we designed a middleware to fulfill the autonomic or organic computing demands of self-configuration, self-optimization, self-healing, and self-protection. We investigated self-configuration by a social-cooperative behavior, self-optimization by an artificial hormone-based approach, self-healing by dependability approaches, and self-protection by computer

\footnotesize{2We coined the word probody because there is no handy notion in immunology for the item that is required in our artificial immune system.
immunology. The high-light of our system are the Smart Doorplates that serve as test and presentation platform. On the Smart Doorplates we validated the functionalities of an organic computing middleware and implemented several services on basis of mobile agents.

REFERENCES


CONCLUSION


