DISSERTATION

Mobility-Aware Distributed Computing in Shared Data Spaces

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines
Doktors der technischen Wissenschaften unter der Leitung von

O.Univ.Prof. Dr. Günter Haring
Institut für Distributed and Multimedia Systems (396)

eingereicht an der Technischen Universität Wien,
Fakultät für Informatik

von

Dipl.-Ing. Karin Anna Hummel
Matr.Nr. 8900972
1030 Wien, Barmherzigengasse 17/4/32

Wien, im Februar 2005

Karin Hummel

Mobilitätsbasierte verteilte Berechnung in gemeinsamen Datenräumen

Kurzfassung

Mobile Computing kann als eine natürliche Weiterentwicklung verteilten Rechnens gesehen werden, die durch drahtlose Netzwerke, portable Geräte und Unterstützung von Personenmobilität, wie zum Beispiel Tracking Technologien, ermöglicht wird. Die Integration mobiler Komponenten in stationäre Systeme stellt jedoch aufgrund begrenzter Leistungsfähigkeit mobiler Geräte, sowie der sich häufig ändernden drahtlosen Netzwerkqualität, eine Herausforderung dar.

Software Adaptierbarkeit für mobile Rechnerarchitekturen, schnelles Auffinden verfügbarer Dienste, so wie die Unterstützung für asynchrone Kommunikationsprotokolle, wie sie von *Data Space* basierten Technologien angeboten werden, sind wesentliche Anforderungen. Oft werden mobile Geräte mittels Proxy Technologie oder leichtgewichtiger Protokolle integriert, wobei die wachsende Leistungsfähigkeit mobiler Geräte zunehmend die Integration als vollwertige Teilnehmer ermöglicht.

Der Beitrag dieser Dissertation zur Forschung im Bereich Mobile Computing liegt in der Einführung einer mobilitätsbasierten, fehlertoleranten Software Schicht aufbauend auf Data Space Technologie. Die Kenntnis über das Mobilitätsverhalten gründet sich auf Beobachtungen der Zustände und der Zustandswechsel im drahtlosen Netzwerk, sowie auf Vorhersage des nächsten Zustandes mittels Mobilitätsmodellen. Reaktive-, wie auch proaktive Fehlertoleranzmechanismen verwenden Kopien von gemeinsamen Daten.

Die Anwendbarkeit des Konzepts wird mittels einer Referenzimplementierung aufbauend auf der Middleware CORSO demonstriert. Die Vorhersage basiert auf zwei Modellen, einem Markov Modell zweiter Ordnung und einem personenbezogenen terminplan-basierten Modell, das zum Beispiel aus Kalendereinträgen extrahiert werden kann.

Experimente in einem WLAN Hotspot Bereich zeigen die Auswirkungen bewegungsbasierter Koordinierungsfehler und evaluieren die Fehlertoleranzmechanismen. Die Vielfalt an möglichen Koordinationsszenarien wird mittels vier bezüglich zeitlicher und referentieller Kopplung der beteiligten Prozesse unterschiedlich zu klassifizierenden Koordinationsmuster reduziert. Die Resultate zeigen die von synchroner Kommunikation bedingten Grenzen und den Nutzen des Ansatzes.

Mobility-Aware Distributed Computing in Shared Data Spaces

Abstract

Mobile computing is a natural further development of distributed computing enabled by wireless networking technologies, portable devices, and means for supporting personal mobility based on, for example, tracking technologies. However, the integration of mobile technologies into stationary distributed systems causes several challenges due to mobile device limitations and changing wireless network connectivity.

Many research efforts focus on software adaptations for mobile computer architectures, on fast service discovery, and on lightweight and asynchronous coordination protocols as provided by *data space-based* technologies. Mobile devices are commonly integrated by means of proxy technologies and lightweight protocols. As mobile computing devices become more and more powerful, integrating these devices as equivalent participants within distributed architectures becomes feasible.

This thesis contributes to the research field of mobile computing by proposing a mobility-aware fault-tolerant coordination layer on top of data space-based middleware. Mobility-awareness addresses observation of current wireless link states and prediction of future wireless link states and retention periods based on mobility models. Hence, both reactive and proactive mechanisms are proposed to tolerate weak network conditions and disconnections by means of local replication of the shared data space.

An extensible reference implementation based on the shared object space middleware CORSO demonstrates the feasibility of the approach. For prediction purpose, a second order Markov model for modeling continuous movement in terms of direction and a schedule-centered model based on, for example, a person's calendar entries, are used as reference mobility models.

Experiments are carried out to investigate coordination failures caused by moving in a wireless LAN hotspot area and to analyze fault tolerance mechanisms. Four well known coordination patterns are used as representatives for a manifold of coordination scenarios classified according to the coupling characteristics of processes in terms of time and reference. The results of the experiments are used to investigate possible achievements and inevitable limits of the mobility-aware approach for each coordination pattern.

Danksagung

Diese Arbeit entstand im Rahmen meiner wissenschaftlichen Tätigkeit am Institut für Distributed and Multimedia Systems, Fakultät für Informatik, der Universität Wien.

Besonders danke ich meinem Betreuer, Prof. Günter Haring, der mir die Forschungstätigkeit am Institut ermöglichte und mich mit hilfreichen Anregungen vor allem in der schwierigen Endphase der Arbeit unterstützte.

Weiters möchte ich die Gelegenheit nutzen, mich bei meinen Kollegen und Kolleginnen am Institut für Distributed and Multimedia Systems sehr für ihre Hilfsbereitschaft und auch Freundschaft zu bedanken. Besonderer Dank gebührt hier Helmut Hlavacs für zahlreiche fachliche Diskussionen, die wir meist spontan führen konnten, aber auch für das sorgfältige Korrekturlesen dieser Arbeit.

Ganz herzlich bedanke ich mich bei Frau Prof. Eva Kühn für die engagierte Begutachtung dieser Dissertation, insbesonders aber auch für unsere Gespräche, die halfen, den Blick auf offene Fragestellungen im Bereich des Space Based Computing zu richten.

An dieser Stelle möchte ich auch meinen Freunden und Freundinnen, und meinen Verwandten Dank sagen, die mich geduldig während einer Zeit begleiteten, in der ich leider sehr wenig davon für sie übrig hatte. Meinen Eltern und meiner Schwester, Susanna Militky, danke ich besonders für ihr Verständnis. Vor allen anderen danke ich Michaela Holzer für zahlreiche bestärkende emails und Gespräche.

Am meisten danke ich Thomas Galla für seine oft heldenhafte Unterstützung auf so vielen Ebenen. Das fachliche Interesse und seine kritischen, aber auch bestätigenden Kommentare, sowie seine Bereitschaft, diese Arbeit Korrektur zu lesen, halfen sehr, die Qualität zu verbessern. Mehr noch aber, möchte ich ihm hier für sein Vertrauen in mich und seine Liebe danken.

Contents

Chapter 1

Introduction

Distributed systems become more and more enriched by mobile computers and mobile appliances. Those devices allow to connect to Web-services, ad-hoc collaborative applications, and legacy systems. Although a manifold of mobile computers exist, the common ground of mobile computing lies in the dynamics due to *movement*. This thesis studies the effects of mobility applied to one of the major characteristics of distributed computing, that is, *coordination*.

1.1 Mobile and Wireless Computing Fields

Mobile computing can be seen as a natural extension of distributed computing simply because the participating distributed computers are portable and, thus, exhibit the potential to be moved. Due to the need of user mobility, devices used differ in weight, size, and technical footprints. For example, while smart phones are on the lower end of the scale, Personal Digital Assistants (PDAs) exhibit more processing power, more main memory available, and larger display sizes. Notebooks are high-end mobile devices often capable of replacing desktop computers.

Trends in mobile computing are most interesting in market research concerning both preferred wireless network technologies and mobile devices. Due to an economy research by TNS Infratest on behalf of the German government [Gra04], mobile communication and mobile computing are expected to grow significantly. In April 2003, worldwide 40 percent of mobile phones in operation have already been Internet enabled. WLAN is currently the technology that is more frequently used than UMTS. Different indicators make the prognosis reasonable, that WLAN will stay important during the next years. For examples, public WLAN hotspot installations will grow as depicted by Figure 1.1(a), while the number of mobile devices sold which support WLAN access will significantly in-

crease (Figure 1.1(b)). WLAN enabled smart phones are expected to be a major technology for the next years, since they are able to combine the functionality of PDAs and mobile phones.

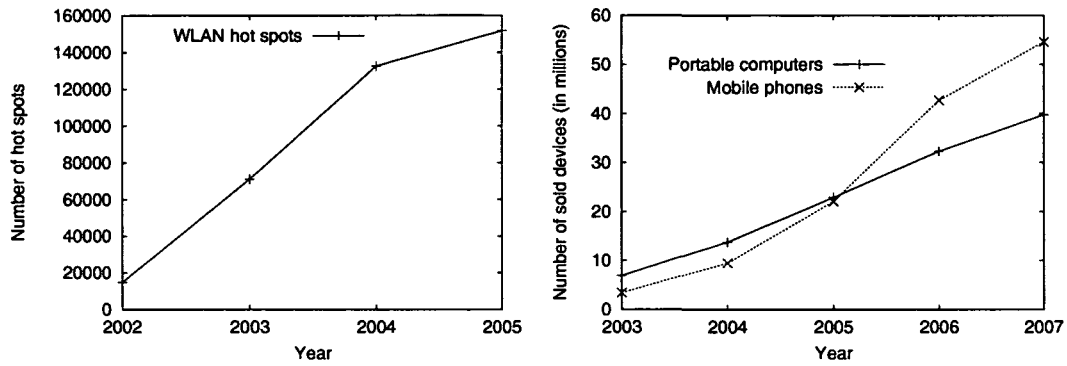


Figure 1.1: (a) Number of public WLAN hot spots worldwide and (b) growth rate of WLAN enabled mobile device markets worldwide - year 2003 and further are prognosis values (studies have been first published in summer 2003) [Gra04]

As a consequence of the widespread availability of networked mobile devices and wireless networks, during the last decade, mobile computing has already been widely introduced in the following fields:

M-Work. Since field technicians, medical personnel in hospitals, and managers on business trips are mobile, here mobile devices and wireless networks provide access to computation support, enterprise specific applications, and Internet services.

M-Commerce. Wireless networks and protocols like WAP/WML[1] together with mobile devices enable users to connect to Web-shops, newspapers, and Web-based booking services. In the field of mobile telecommunications, SMS services can be used to buy products, like soft drinks or railway tickets, while being on the move.

M-Learning. Education is a major sector where notebooks are widely used. In particular, university campuses are more and more enriched by WLAN access points [Sho01]. Going one step further, context-aware learning applications can be used to enable collaborative learning or learning activity selection and adaptation based on the current situation. Various applied research has been carried out at the Institute of Computer Science and

[1] http://www.w3c.org/

Business Informatics at the University of Vienna in the field of m-learning. For example, a learning unit recommendation prototype has been proposed which is based on augmenting learning units by XML meta-data. This meta-data allows to describe, for example, context requirements of a learning unit [Hum03].

Tracking students interactions with both digital and physical learning resources has been proposed in [Hum04b]. Here, location and proximity awareness tracking is based on *Radio Frequency Identification (RFID)* technology. Figure 1.2(a) shows the learner user interface of the first prototype and Figure 1.2(b) shows the concept of the tracking prototype.

M-Entertainment. Mobile location based games running on smart phones or PDAs, mobile e-city guides, e-museum guides, as well as movies streamed to or running on mobile devices are examples of mobile entertainment. An examples of such a gaming prototype situated in Venice is presented by Bellotti et al. [Bel03b].

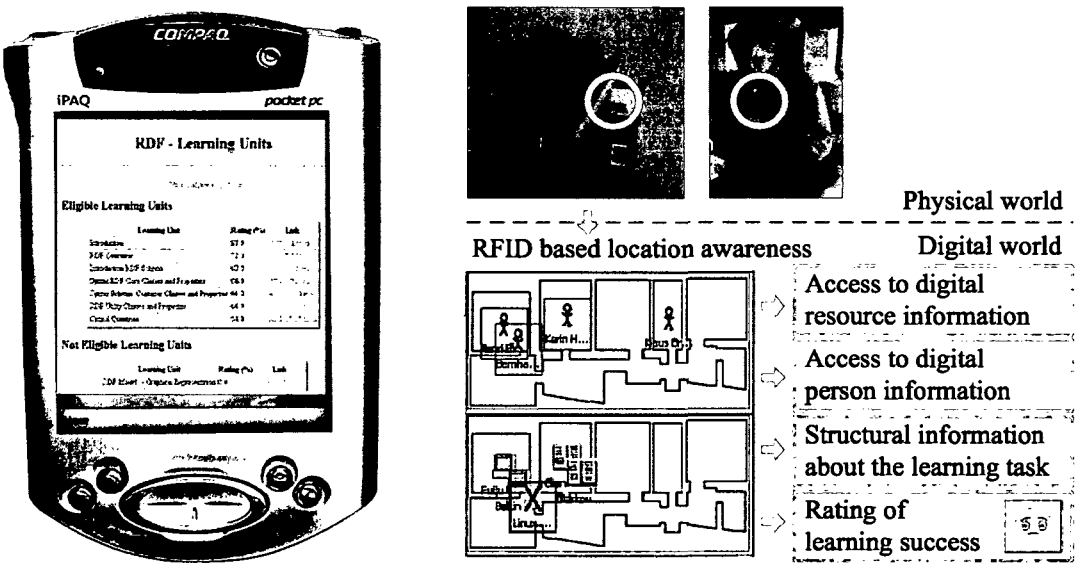


Figure 1.2: (a) M-learning recommendation prototype and (b) prototype for tracking RFID-tagged persons and resources in order to generate physical learning traces

In addition to everyday life applications, progress in mobile computing and in mobility modeling is needed in the following current research fields:

Ubiquitous Computing. In ubiquitous computing environments, persons are envisioned as being surrounded by an ambient computational environment.

Personal mobility and, if applicable, terminal mobility require support for changing contexts due to movement. For example, digital mobile services might follow a person's physical movement (*code on demand* or *mobile agents* [Zac02b] and adapt to the current context).

Distributed Computing. Traditional distributed computing services are provided by general purpose middleware systems. In addition to stationary computers, these systems consist more and more of mobile devices, like PDAs and notebooks or even smaller appliances. Compared to stationary computers, mobile devices show different behavior mainly in terms of *availability* and *processing and memory power*. Without a model for movement and a concept for maintaining connections to moving devices and to devices with small technical footprints, only a few application areas might be covered. CORSO mobile, for example, enables the connection of small mobile devices via Java&Co, that is, an API which enables processes to connect to a remote shared data space [Küh01]. Recently, also mobile and wireless computational grid systems are proposed [Kur04, McK04]. Here, the management of mobile nodes used as, for example, computing resources, and their dynamics in terms of availability are of major interest.

Wireless Networking and Routing. When movement has to be considered, wireless networks have to face some challenges in terms of channel allocation, handover, and borderline problems between neighboring cells. Here, models of movement are used for future movement prediction purposes in order to avoid network management overload. Furthermore, when evaluating new routing or management algorithms, mobility models are used for simulation purposes [dW03, Cay02].

1.2 Problem Definition and Contribution of this Thesis

Among the mobility related research topics identified for distributed computing, coordination of processes executing on mobile devices is a major challenge. Traditionally, coordination between distributed processes is analyzed, modeled, and implemented assuming stationary network components. In case of mobile distributed systems, adaptations are required which allow to handle components that become unavailable frequently or exhibit significant wireless link quality changes over time. From a coordination scenario's perspective, these characteristics can be regarded as faults. Consequently, fault tolerance mechanisms can be used to assure ongoing or degraded operation. Such assurance is very important, for example, in case *Service Level Agreements (SLA)* have to be met by a service provider.

Early related works, like the resource-centric work proposed by Schill et al. [Sch95], have already identified major means for mobility support, that are, emulation of resources, caching, and use of (other) local resources based on the *Distributed Computing Environment (DCE)*. By using middleware approaches which provide stateful communication and coordination resources, like proposed by CORSO [Küh01], participants may reconnect and retrieve information stored while they have been disconnected. However, times of disconnections are commonly not supported. More recent works address this issue by working on copies, which are, (i) creation of the copy and (ii) synchronization when reconnecting. For example, Mascolo et al. [Mas02b] propose versioning as a means for synchronization.

These reactive mechanisms are not sufficient for activities which have to be finished before the network condition worsens or the connection breaks. Hence, very recently, proactive mechanisms are proposed which use uncertainty modeling and reasoning about mobile behavior, like the work of Burcea et al. [Bur04] for pre-fetching of events. However, there is a lack of a common understanding of the major characteristics of mobility modeling, mobility caused failures, and their consequences.

The work presented in this thesis aims at providing such characteristics for mobile behavior, modeling mobility caused failures, investigating their consequences, and deriving applicable reactive and proactive fault tolerance mechanisms. Hereby, this thesis focuses on wireless networks characterized by changing link qualities and on high-end mobile devices, like notebooks and tablet PCs, which already allow mobile components to be full participants in a distributed system. Due to the superior support provided by shared data space approaches, which originate from the *Linda tuple space* model [Gel85, Gel92], the approach proposed is based on this paradigm. The consequences of movement for distributed coordination is studied related to the following critical issues:

Availability. Since ubiquitous and seamless network support cannot be assumed for mobile devices, disconnections are *planned failures*. Concepts of mainly asynchronous communication and persistent message storage offer superior support for these failures. When comparing the two different paradigms used for distributed computing, that are *message passing* and *virtual shared memory*, approaches implementing the virtual shared memory paradigm are more promising. These distributed systems use the abstraction of a shared data space, where messages are stored at and retrieved from at arbitrary times instead of transferring messages between processes [Küh98b, Gel85]. However, in case the virtually shared space is distributed between mobile computers, parts of the shared space may become unavailable in case one mobile component disconnects. A definition of the resulting failures, their consequences, and feasible fault tolerance mechanisms are required.

Service Provisioning over Degraded Link Quality. Roaming within a wireless network may lead to link degradation due to noise disturbances, shielding, and reduction of signal strength. In order to hide link degradation from the user perception, compensation actions are required. For example, network traffic or synchronization operation may be postponed until network quality gets better. Here, similar to disconnection faults, a mapping between link quality and resulting adaptations of the distributed system or the application is required.

Working on Copies. In order to support operations on shared data while being disconnected, processes on mobile devices will often operate on copies and cached data. Accuracy while creating the copy, efficient synchronization, and data lock management have been identified as key challenges [Imi94] and are still addressed by recent middleware approaches (see, for example [Cap03]). Consistency assurance itself has to be addressed as well as the question how and when to initiate corresponding operations. The latter is one of the most challenging issues addressed by this thesis.

Consequences of Movement on Coordination Tasks. Since distributed coordination tasks exhibit different degrees of communication load and style, mobility related failures are expected to differ depending on the type of coordination operations carried out. In order to discuss these mobility related consequences systematically, coordination tasks have to be classified and studied accordingly, which is an open issue up to now.

Figure 1.3 depicts the approach proposed for evaluating the consequences of mobility on distributed coordination and for compensation of failures caused by mobility. First, in order to study the effects of mobility on distributed coordination, the manifold of possible scenarios is reduced by means of *coordination patterns*. Second, the core of the supporting fault tolerance mechanisms proposed is designed as a modular *mobility-aware coordination layer* used to advance the concept of the shared data space. Finally, experimental evaluation is used to show the improvements achieved by the mobility-aware concept and the limits of the approach for different coordination patterns. These experiments are based on both physical measures and simulation results. The major contributions and methods applied are detailed as follows:

Coordination Patterns. Coordination patterns are software design patterns which provide a useful abstraction from specific distributed computing problems. The modeling approach introduced extends the descriptive template commonly used for design patterns [Gam95], pp. 6–8. *UML activity diagrams* are applied to visualize coordination patterns and to derive *sequences of data space primitives* describing the coordination behavior of the

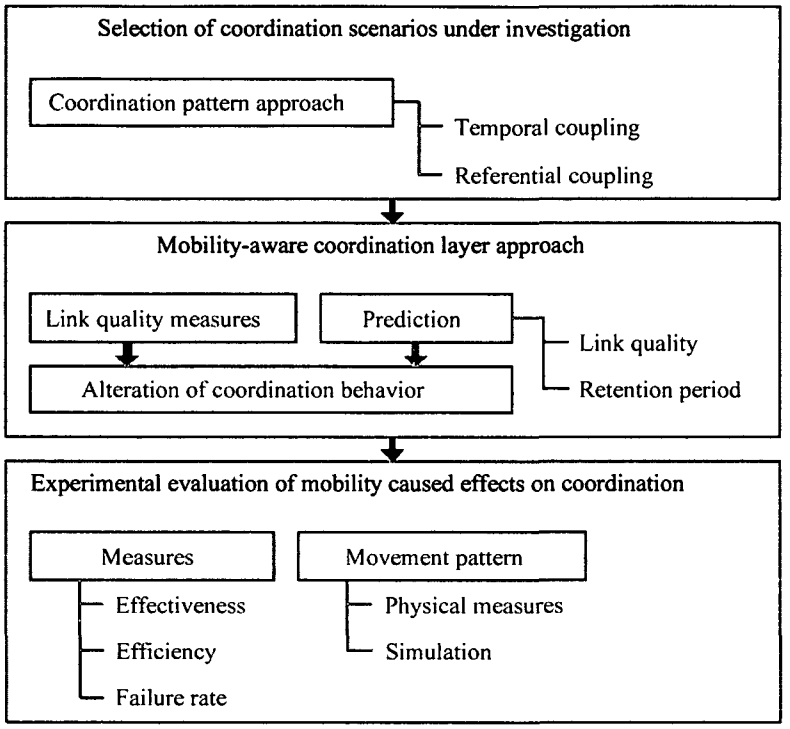


Figure 1.3: Overview of the approach proposed

process types. Additionally, *measures* are defined which allow to quantify coordination patterns in terms of temporal and referential coupling. These two dimensions of coupling are based on a commonly used taxonomy for distributed processing [Tan02, Cab00b], which defines four classes. Coupling of processes is a major limiting factor for the benefits achievable by the mobility-aware coordination layer.[2] The modeling approach is applied to eight significant coordination pattern examples, which are classified accordingly. Among these patterns, four class representatives are selected, which are implemented for comparison reason.

Mobility-Aware Coordination. Based on a fault-hypothesis, the mobility-aware coordination layer introduces and implements the concept of separating a *local copy* from the *global space* in order to provide fault tolerance. Copy, release, and synchronize operations are used to assure consistency between the spaces. Since these operations require invocation in advance, *proactive mechanisms* are proposed. Proactive behavior is achieved by means of *mobility models* which are used to predict future link states

[2]For example, two temporally coupled processes rely on synchronous communication and cannot benefit from working on copies in times of disconnections.

and retention periods by comparing user history traces against mobility patterns. This concept is applied to two different mobility models. First, a second-order Markov model is assumed for wireless link state prediction and a first-order Markov model is used for predicting the next retention period. Additionally, a mobility model is defined which makes use of person centric information, like, for example, a personal time schedule, in order to predict the next link state. The prototypical implementation of the mobility-aware coordination layer is based on evaluating the Signal to Noise Ratio (SNR) based on WLAN 802.11b, the space based middleware CORSO [Küh01, TEC04] (and the Java&Co API), and the Java programming language.[3]

Experimental Results. Experiments based on *physical roaming* and *simulation* are carried out in order to evaluate the mobility caused effects in terms of effectiveness, efficiency, and failure rate. The experiments are applied to all coordination pattern examples which allows to compare the results in terms of *temporal coupling* and *referential coupling*. Additionally, the experiments are used in order to show the benefits and limits of the mobility-aware coordination layer and, in addition, of replication of processes.

Structure of this Thesis

The remainder of this thesis is structured as follows:

Related Work. Section 3 surveys middleware approaches, mobile agent systems, distributed data management approaches, and selected context-aware application approaches. The focus of the comparison and the discussion is set on mobility support and proactive fault tolerance mechanisms.

Coordination Patterns. In Section 4, a modeling approach based on the shared data space approach, a classification scheme in terms of temporal and referential coupling, and a detailed description of coordination patterns is presented. The specification of eight coordination patterns intends to be a reference work for the interested reader.

Mobility-Aware Space Based Computing. Section 5 introduces the concept of mobility-awareness to data space based computing. Here, the fault-hypothesis and the mobility models used for prediction purpose are described. The state machine and the core algorithms proposed are presented in detail.

[3]CORSO originates from the research carried out under the guidance of Prof. Eva Kühn at the Vienna University of Technology.

Application. The mobility-aware concept is applied to the space based middleware CORSO in Section 6. Implementation specific decisions, CORSO features, the modular concept of the mobility-aware coordination layer, as well as data structures and operations used are discussed.

Evaluation. Section 7 describes the experimental setup and presents the results achieved by studying representative coordination scenarios while roaming. The experiments are carried out either by physical roaming or, more extensively, based on a distributed simulator which generates mobility traces.

The conclusions derived in Section 8 complete this thesis.

Chapter 2

Concepts, Terms, and Definitions

This chapter presents several terms and definitions related to the fields of mobile computing, fault-tolerant computing, and coordination in distributed systems. Furthermore, basic concepts and technologies are summarized in order to provide a brief introduction to the main mobile distributed computing issues.

2.1 Mobile Computing

Coulouris et al. [Cou01], p. 6, define *mobile computing* as the performance of computing tasks while the users are on the move, or visiting places other than their usual environment. Campadello et al. [Cam99] extend this definition by differentiating between roaming devices, users, and software. Respectively, mobility is classified into *personal mobility*, *terminal mobility*, and *code mobility*.

Mobile code is software that can be sent from one computer to another and can be executed on the destination computer, like, for example, Java applets. In case mobile code migrates autonomously, it is termed *mobile agent* [Whi97, Pic98, Cou01]. In case mobile agents save their current execution state before migrating in order to continue execution at this saved program state, they are said to perform *strong migration*. Otherwise they are said to perform *weak migration*. Recently, mobile code and mobile agents have been accepted widely as a new coordination paradigm. They are advantageous, whenever it requires less network load to transfer the code to the data than to communicate between remote processes.

Following these definitions, here, mobile computing is defined as the execution of computing tasks while a person, a device, or code is on the move or visiting places other than the usual environment.

2.1.1 Wireless and Mobile Network Protocols

Mobile computing targets all issues concerning mobile devices and their network connections to other devices and computers. Thus, mobile computing relies on a supporting network infrastructure. Although related, mobile computing is not equivalent to wireless computing, that is, network computing over a wireless network [Tan03], p. 10. However, since wireless networks provide easy access and an extended freedom of movement, they are commonly preferred in the field of mobile computing.

Wireless computer networks either define at least the physical layer and the data link layer, or use a wireless infrastructure for data communication services. For example, General Packet Radio Service (GPRS) utilizes the Global System for Mobile Communication (GSM), and 3G (or UMTS) unite data and telecommunication networking. In wireless Local Area Networks (LANs), mobile devices may connect via radio technologies, like Wireless LAN (WLAN, Wi-Fi), which has been standardized by the working group 11 of the IEEE 802 standardization community [IEE99], or Bluetooth. The physical layer and the data link layer of Bluetooth are standardized in IEEE 802.15 [IEE02, Tan03]. Bluetooth is most frequently used in Wireless Personal Area Networks (WPANs) to connect different devices with one another in a point-to-point manner. Additionally, wireless data transmission in the infrared frequency band is here used, which requires a line-of-sight during operation, like IrDA.[1]

Using WLAN, computers may either operate in *ad-hoc mode* or *infrastructure mode*. While in ad-hoc mode a set of equivalent stations connect spontaneously to each other, in infrastructure mode an access point is used to bridge between the wireless LAN and the wired LAN. The access to the wireless medium is either managed by the Carrier Sensing Multiple Access/Collision Avoidance (CSMA/CA) strategy when operating in decentralized Distributed Coordination Function (DCF) mode, or by the centralized Point Coordination Function (PCF) mode. Using CSMA/CA, the station which wants to send first senses the channel whether it is idle. In case it is not, it waits for the duration of back-off interval prior to starting to sense again. Channel sensing may be either based on physical signal sensing or virtual sensing, where specific messages are used to detect whether a channel is idle. Using PCF, the access point polls each stations and grants sending permission in order to prohibit collisions. A brief introduction to WLAN can be found in Tanenbaum [Tan03], pp. 292–302.

At first WLAN operated with maximal data transfer rates (DTR) of 1 or 2 Mbit/s. Newer versions provide higher maximum DTR as shown in Table 2.1, but are still working far below the average speed of todays wired LANs (100 Mbit/s). The standards IEEE 802.11a/b/g[2] all focus on encoding and use two

[1] http://www.irda.org/

[2] For an overview, see, for example, http://en.wikipedia.org/wiki/802.11/.

Standard	max. DTR	Frequency Band	Modulation Method
802.11a	up to 54 Mbps	5-GHz ISM band	OFDM
802.11b	up to 11 Mbps	2.4-GHz	DSSS, HR-DSSS
802.11g	up to 54 Mbps	2.4-GHz	DSSS, HR-DSSS, OFDM

Table 2.1: IEEE 802.11x main standards

different modulation methods, that is, Orthogonal Frequency Division Multiplexing (OFDM), and High Rate Direct Sequence Spread Spectrum (HR-DSSS). In OFDM, frequency hopping is used for modulation, which makes this scheme robust with respect to radio interference. HR-DSSS uses either code shift modulation or Walsh/Hadamard codes on the whole spectrum. The third modulation method in WLAN works with diffused transmissions in the infrared spectrum, but since it offers only low bandwidth, this solution has not been focused on recently. The other standards (802.11c-f,h-j,n) focus on service enhancements, extensions and corrections [Tan03].

When supporting mobile devices (here, also termed *mobile hosts*) on the network layer, network address management is a major issue. In the Internet, network layer mobility mainly suffers from the IPv4 addressing and routing strategy. When addressing hosts, or, more general, network interfaces, the topological structure of the network is explicitly used. An IP address is built up by a prefix referring to the network of the host, followed by a host's address. Based on a dynamically changing and best-effort strategy, IP datagrams are routed to the network stated by the receiver host's IP address. As a consequence, in case a mobile device moves from one network to another, the network prefix of its IP address has to be changed. In case the host is not expected to be available using its old IP address, a new address may be assigned manually or automatically using, for example, the Dynamic Host Configuration Protocol (DHCP) [Dro97]. Mobile IP has been designed in order to transparently relay IP data transfer to the mobile host's new IP address. The protocol is based on the utilization of a *home agent*, which acts as a proxy for the moving host in the home network by tunneling IP packets to the visited network (foreign network). For further reading see Perkins [Per96, Per98], or an introductory by Hummel et al. [Hum04c].

2.1.2 Network Quality of Service in Wireless Networks

In a distributed system, the performance of the communication channels is important for the overall performance of the system. Coulouris et al. [Cou01], p. 51, describe the main performance characteristics in terms of latency, jitter, and bandwidth. *Latency*, is defined as the delay between the sending of a message

initiated by an application process and the receipt of this message by the receiver process. *Jitter* is the variation in delay for delivering a series of messages. *Bandwidth* is used synonymously to *Data Transfer Rate (DTR)* by Coulouris et al. [Cou01], pp. 49–50. It is defined as the maximum amount of data which can be transmitted in a given time interval.

In wireless and wired networks, the signal strength determines the maximum DTR. Additionally, for example, thermal noise further reduces the available maximum DTR. Tanenbaum et al. [Tan03], p. 89, describe the relation of maximum DTR briefly, which originates from the works of Nyquist and Shannon. Equation 2.1 shows Shannon's result for a noisy channel, where H denotes the bandwidth (in Hz) and S/N the ratio of signal strength to noise strength *(Signal-to-Noise Ratio (SNR))*. The maximum DTR calculated can be seen as an upper bound for the actually observed DTR on communication channels:

$$maximum\ DTR = H \log_2(1 + \frac{S}{N})\ [bit/s] \tag{2.1}$$

Since the SNR can be sensed without any further probing of the communication channel, it is reasonable to use it as an indicator for link quality and thus for determining the degradation level of possible DTR.

In addition to the DTR, packet losses and delays are further channel characteristics that influence the *end-to-end QoS*. Network protocol adaptations are often required for wireless networks, like proposed by various TCP adaptations for the transport layer ([Tan03], pp. 553–556). In heterogeneous networks which may consist of different wireless and wired networks, end-to-end QoS provisioning architectures have to address different network protocol QoS mechanisms. For example, *Integrated Services (IntServ)* and *Differentiated Services (DiffServ)* allow to reserve resources for different traffic flows and users (see, for example, Tanenbaum et al. [Tan03], pp. 409–415), while *Multiprotocol Label Switching (MPLS)* [Ros01] provides selection of routes (i.e. switching functions) by means of header labels independent from data link and network layer. Recent work by Gao et al. [Gao04] propose a novel QoS provisioning architecture which allows *hyper handover*. Here, multiple network protocol layer services are integrated during changes in terms of access technology, terminals used, application characteristics, and administration domains.

2.1.3 Mobility Models

A *mobility model* defines a notation and a description of location sequences describing the motion of an entity in space and time. Usually, additional parameters of interest, like direction, velocity, acceleration, retention period, and a time stamp are included. From such a model, it is possible to extract these sequences (*mobility patterns*) and to compare them to *mobility traces*, that are, concrete

movement observations from roaming entities. The more mobility models provide accurate, scalable, adaptive, and inexpensive algorithms for comparing mobility patterns against traces, the more they are applicable to different scenarios. For example, the extraction of every day motion patterns requires other temporal and spatial granularities, than the extraction of motion patterns at a conference venue.

According to the notion introduced by Cheng et al. [Che03c], let A be a set of possible locations (for simplicity without any additional parameters). A location sequence is also termed *movement history* H_n, where

$$H_n = \langle X_1 = a_1, ...X_n = a_n \rangle , \tag{2.2}$$

where X_i denotes a random variable and each $a_i \in A$. $P(X_i = a_i)$ denotes the probability that the random variable takes value a_i and $\hat{P}(X_i = a_i)$ denotes an estimation of this probability. The events are commonly assumed to occur at equally spaced time intervals.

2.2 Fault Tolerance and Availability

More than in stationary distributed systems, in mobile computing systems, communication breaks and decreased DTR are not an exception, but the usual case. Thus, the distributed mobile system has to stay operational in the presence of such faults.

2.2.1 Fault-Error-Failure Definitions

According to Laprie et al. [Lap92], a *fault* is an adjudged or hypothesized cause of an error. An *error*, that is, the manifestation of a fault, often leads to a failure of the system. A *failure* is defined as a system state, in which the delivered service no longer complies with its specification. In distributed systems, the failure of one sub-system may lead to a fault in another system recursively. A *fault-hypothesis* describes the possible faults, errors, and failures in a system. Based on the fault-hypothesis, system degradation states may be derived and fault tolerance mechanism may be specified.

Figure 2.1 shows a classification of failures widely used in embedded distributed systems, like described by Kopetz, [Kop97], pp. 120–123. The *nature of failures* distinguishes between the *time* and the *value* domain. For example, messages arriving too late are timing failures, while messages containing wrong data are value failures. The *perception of a failure* may be assumed to be *consistent* for all partners, like used in atomic group multicast protocols, or to be *inconsistent*. In terms of *effect*, failures can be *malign* and *benign*. Since mobile

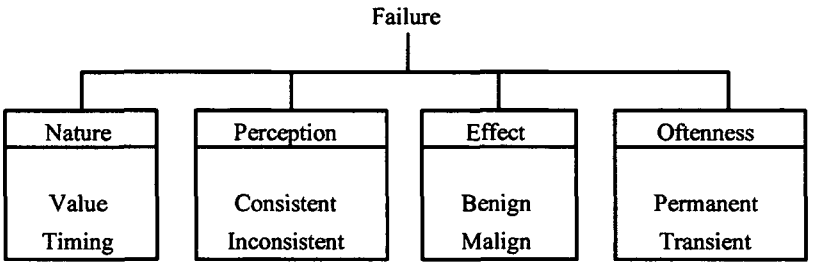


Figure 2.1: Failure classification

computing does not target hard real-time systems, this property is not addressed in the presented work. Finally, in terms of *occurrence*, failures can persist *permanently* or be observed *transiently*. After a permanent failure, which is a special case of a *single failure*, the system ceases to provide a service until it is repaired. Otherwise, in case a failure occurs and vanishes again while the system remains operating the failure is termed *transient failure*. In case a transient failure occurs frequently, it is termed *intermittent failure*.

Kopetz furthermore distinguishes between two cases classified as consistent failures, namely *fail-silent*, where a component does not deliver any output in case of an error, and *fail-consistent*, where a component delivers the same (wrong) value to other components. Inconsistent failures are often termed *Byzantine failures* (or *malicious failures*).

A fault-tolerant subsystem (or unit) consisting of redundant components has to reach agreement in spite of k faulty components. Caused by the semantics of the components' failure modes, such a unit has to consist of a minimum number of components, that is:

- k+1 in case of fail-silent components,
- 2k+1 in case of fail-consistent components,
- 3k+1 in case of Byzantine components.

In case of k fail-silent components, only one correct component that delivers a correct message is necessary. If k components fail consistently, k+1 components that send correct values are necessary to select the correct value by means of majority voting. In case of Byzantine components, k+1 components are not enough to tolerate k inconsistent failures because they are not perceived similarly. It has been shown, that this problem is solvable for a minimum of 3k+1 components which perform several communication rounds [Pea80].

In terms of dependence, failures may either be *independent* or *related.* In the related case, failures may either be coincident failures (that is, they occur because of the same input, but are not related), simultaneous failures, or sequential failures.

2.2.2 Failures in Distributed Systems

In distributed systems, failures of computing entities are perceived as communication failures. Messages may be lost, delayed, contain wrong data values (for example, may be corrupted due to bit-flips), received multiple times, or received in wrong order. As a consequence, the fault hypothesis may be reduced and simplified to those perceived communication failures and the testing of fault tolerance mechanisms can be based on a reduced test set.

In mobile distributed systems, communication failures are mainly caused by network problems, which occur depending on the mobility behavior and network coverage. Thus, the fault-hypothesis here may be used to define degradation states caused by roaming in areas where bad network connection is available. Table 2.2 presents communication failures based on the terminology presented by Tanenbaum et al. [Tan02], pp. 364–365, in terms of *failure perception* and *error* (or failure cause).

Perception	**Error**
Timing	Message Loss or Delay, Sender's or Receiver's Omission, or Crash
Multiplication	Network Communication Protocol: Resend, Buffer Error
Value	Bit-Flips during Transmission, Sender's Response
Ordering	Network Communication Protocol Deficiency or Error

Table 2.2: Communication failures

Usually, reliable network protocols, like the Transmission Control Protocol (TCP, initially defined in RFC 793 [Pos81]), provide means in order to avoid and tolerate some of these faults, for example, ordering or message loss. In case of using an unreliable transport protocol, like User Datagram Protocol (UDP, defined by RFC 768 [Pos80]), the middleware or application process is responsible to add fault tolerance mechanisms.

2.2.3 Reliability and Availability

The *reliability R(t)* of a system is defined as the ability to perform its function (according to the specification) for a specified time, when required. It is mea-

sured as a probability that changes over time (for a detailed introduction see, for example, Leitch [Lei95]). If *f(u)* is assumed for the distribution of failure times, it can be used to calculate the reliability by the following formula:

$$R(t) = \int_t^\infty f(u)du \tag{2.3}$$

Whenever a constant failure rate can be assumed (that is, $f(u) = \lambda$), R(t) can be calculated as follows:

$$R(t) = e^{-\lambda t}. \tag{2.4}$$

In mobile distributed systems, failures due to connection loss or decreased DTR, are intermittent failures. A simple model considering both failures and re-establishment of the system's services can be seen as a *birth-and-death* process (see Allen [All90], pp. 121–129). Figure 2.2 shows a simple Markov model of a system with two states, an operational state (*0*), and an error state (*1*). λ denotes the constant failure rate and μ denotes the constant repair rate.

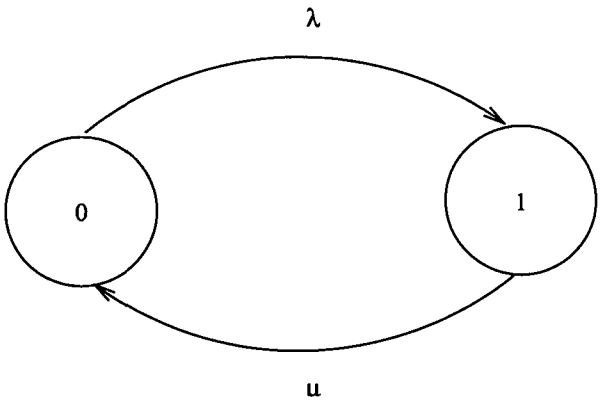


Figure 2.2: Markov model of a repairable system

Let $P_0(t)$ denote the probability that the system is in state 0 at time t and $P_1(t)$ denote the probability that the system is in state 1 at time t. Under the assumption that the system is operational in the initial state, in this system, the following conditions are given:

$$P_0(0) = 1,\ P_1(0) = 0,\ \text{and} \tag{2.5}$$

$$\forall t : P_0(t) + P_1(t) = 1. \tag{2.6}$$

The dynamics of the system can be described as differential equations (see, for example, the description in Allen [All90]):

$$\frac{dP_0(t)}{dt} = -\lambda P_0(t) + \mu P_1(t),\ \frac{dP_1(t)}{dt} = \lambda P_0(t) - \mu P_1(t), \tag{2.7}$$

and by substitution, the equation becomes

$$\frac{dP_0(t)}{dt} = \mu - (\lambda + \mu)P_0(t). \tag{2.8}$$

By transformation and elementary differential equation theory (for details see, for example [Heu89, Heu88]), it can be derived that

$$P_0(t) = \frac{\mu}{\lambda + \mu} + (P_0(0) - \frac{\mu}{\lambda + \mu})e^{-(\lambda+\mu)t}. \tag{2.9}$$

For $t \to \infty$, it yields:

$$\lim_{t\to\infty} P_0(t) = \lim_{t\to\infty} \frac{\mu}{\lambda + \mu} + (P_0(0) - \frac{\mu}{\lambda + \mu})e^{-(\lambda+\mu)t} = \frac{\mu}{\lambda + \mu}. \tag{2.10}$$

By symmetry, it can be applied accordingly that

$$\lim_{t\to\infty} P_1(t) = \lim_{t\to\infty} \frac{\lambda}{\lambda + \mu} + (P_1(0) - \frac{\lambda}{\lambda + \mu})e^{-(\lambda+\mu)t} = \frac{\lambda}{\lambda + \mu}. \tag{2.11}$$

The *availability* of a system is defined as the probability $A(t)$ a system operates according to its requirements at time t. Thus, $A(t) = P_0(t)$. A quality measure of a system's overall availability $T_{av}(T)$ (that is, the time, the system remains in operational state during a finite time interval T) can be calculated by integrating $P_0(t)$ (Equation 2.9):

$$T_{av}(T) = \int_0^T P_0(t)dt = \frac{1}{\lambda + \mu}(\mu T - \lambda + \lambda e^{-(\lambda+\mu)T}). \tag{2.12}$$

$\frac{1}{\lambda}$ is often termed *Mean Time To Failure (MTTF)* and $\frac{1}{\mu}$ is often termed *Mean Time To Repair (MTTR)*. The availability of a system at time t may thus be derived by transforming equation (Equation 2.10):

$$\lim_{t\to\infty} P_0(t) = \frac{MTTF}{MTTF + MTTR}. \tag{2.13}$$

2.3 Coordination

In distributed systems, processes, threads, and agents coordinate their actions with one another. In terms of a programming language, coordination is an essential issue, which can be separated from the other computational issues [Gel92, Bus01], like shown in Figure 2.3. The two most commonly used coordination paradigms are *message passing* and *shared data based coordination* (or *virtual shared memory*). Both paradigms offer means for synchronous and asynchronous communication. However, using shared data based coordination,

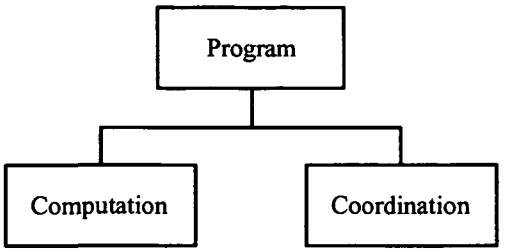


Figure 2.3: Concept of concurrent programming

communication is hidden by the shared data abstraction and the processes are, in principle, uncoupled. Sending and receiving of messages is transformed into reading and writing to shared data.

Busi et al. [Bus01] state the following main issues when designing a coordination language:

- definition of *coordinables*, that is, the active components which are coordinated, like agents or processes;
- selection of *coordination media*, like channels, ports, or shared memory areas;
- design of *coordination rules*, that is, coordination protocols which are represented by sequences of primitives with clear semantics.

2.3.1 The Data Space Model

Based on the Linda Tuple Space introduced by Gelernter et al. [Gel85, Gel92] and techniques from process calculi for concurrency, Busi et al. [Bus01] introduce the following notation for agents operating in a shared Data Space (DS), but it can be used for arbitrary processes as well:

$$P ::= 0 \mid \mu.P \mid \sum_{i=1}^{n} P_i \mid K, \tag{2.14}$$

and

$$\mu ::= out(a) \mid in(a) \mid rd(a) \mid eval(P). \tag{2.15}$$

μ denotes an instance of a possible coordination primitive ($out(a)$, $in(a)$, $rd(a)$, or $eval(P)$, which will be explained later) and P denotes a process in the distributed system. P can either be an empty process, a process that can execute operation μ and the remaining code P, $\sum_{i=1}^{n} P_i$ denotes an *alternative composition*, where the

process may behave like any of the P_i, and K denotes a generic element. This generic element allows to model recursions with this simple notation.

A system configuration consists of two multisets, a multiset of processes *M(Process)* and a multiset of data *M(Data)*. Formally, it can be stated that

$$(PR, DS) \in M(Process) \times M(Data). \tag{2.16}$$

The coordination rules are shown in Table 2.3 (for alternative composition and generic elements, see [Bus01]). The output operation *out(a)* causes an inclusion of item a to the data space, *in(a)* performs a consuming read and *rd(a)* performs a non-consuming read. The semantics of *eval(A)* is the creation of a process (sometimes also termed *spawning*). Both *in*() and *out*() are blocking functions. $\oplus$ denotes a multiset union.

Operation		DS State
$(out(a).P \oplus PR, DS)$	$\longrightarrow$	$(P \oplus PR, DS \oplus a)$
$(in(a).P \oplus PR, DS \oplus a)$	$\longrightarrow$	$(P \oplus PR, DS)$
$(rd(a).P \oplus PR, DS \oplus a)$	$\longrightarrow$	$(P \oplus PR, DS \oplus a)$
$(eval(P').P \oplus PR, DS)$	$\longrightarrow$	$(P' \oplus P \oplus PR, DS)$

Table 2.3: Basic DS coordination model

In the Linda programming language, the non-blocking input operations are termed *inp()* and *rdp()*. In order to be able to specify these operations, the calculus selected needs the specification of a *test-for-absence operation tfa(a)* [Bus01]. It returns *false* if the item a is *not* in the data space, otherwise it blocks:

$$\mu ::= \ldots \mid tfa(a), \tag{2.17}$$

and

$$(tfa(a).P \oplus PR, DS) \longrightarrow (P \oplus PR, DS). \tag{2.18}$$

Thus, $rdp()$ and $inp()$ behave either like their blocking counterparts $in()$ or $rd()$ respectively, or return *false* in case DS does not contain item a. The equations (2.19, 2.20) describe the syntax used. With the $rdp()$ and $inp()$ operations, the calculus becomes Turing-complete:

$$rdp(a)?P_1, P_2 = rd(a).P_1 + tfa(a).P_2, \tag{2.19}$$

$$inp(a)?P_1, P_2 = in(a).P_1 + tfa(a).P_2. \tag{2.20}$$

2.3.2 Distributed Transactions

The operations introduced act autonomously on one data item and thus, concurrency conflicts are implicitly solved. When applying the operations to a set of data items, transactions are a means to ensure consistent data access. Additionally, atomic operations for data-multisets may be used, like the *multiwrite* primitive introduced by T-Spaces [Wyc98, Leh99]. Mechanisms assuring mutual exclusive access to resources similar to Dijkstra's semaphores exhibit the disadvantage of locking a set of distributed processes sharing resources in case a process becomes not available before releasing the lock.

Traditional transactions are based on the ACID properties: *atomicity, consistency, isolation, durability.* While the atomicity property guarantees the execution of all grouped operations or no operation at all, the consistency property guarantees the transfer from one consistent system state into the next. The isolation property states that no transaction should be effected by another transaction and the durability property states that the results of the transaction should be persistent.

This concept may be relaxed, like it is done in CORSO [Küh01, Küh02] by relaxation of the isolation property based on the Flex Transaction model [Buk93]. Here, compensation actions are used to assure consistency in the system in case a sub-transaction fails. Undoing the effects of transactions semantically may be a challenging task in a distributed system in terms of complexity introduced and additional execution time needed.

Chapter 3

Literature Survey

Mobile computing exhibits challenges different from traditional distributed computing which are related to *mobile data management*, *seamless mobile computing*, and adaptations due to limited *mobile device capabilities*.

Imielinski et al. [Imi94] describe the major challenges of mobile computing from a data management perspective. These challenges include (i) *management of location dependent data*, (ii) *disconnections*, (iii) *adaptations of distributed algorithms for mobile hosts*, (iv) *broadcasting over a wireless network*, and (v) *energy efficient data access*. Compared to continuous operation of stationary computers, mobile computing causes more bursts of activities.

Roman et al. [Rom00a] exploit the problem area from a mobile computing perspective. Logical mobility, that is mobility of code in the digital space, and physical mobility are distinguished. In contrast to investigating logical over physical mobility [Zac02b], the notion of *unit of mobility* allows to refer to both classes which simplifies modeling. It is further argued, that algorithms, middleware systems, and applications undergo adaptations caused by mobility. System adaptations envisioned are driven by the need for adding location awareness, for less power consumption, and for using more loosely coupled connectivity patterns.

From a fault tolerance perspective, means for preventing failures of the mobile system are addressed which should allow seamless operation to a high degree. Mascolo et al. [Mas02a, Cap02, Mas04] give a detailed survey on mobility support exhibited by different categories of middleware platforms.

For adapting existing distributed systems to new lightweight mobile devices and for the design of new lightweight distributed computing solutions, communication protocols and APIs have to be re-designed. For example, the Java 2 Micro Edition (J2ME[1]), provides less archives and smaller classes than the standard edition.

[1] http://java.sun.com/j2me/

The term *mobility awareness* refers to the importance of modeling the changes caused by movement in terms of space and time. Although some distributed systems approaches exist which focus on related issues like location or context awareness, only a few approaches use the notion of mobility or a mobility model explicitly for service provisioning. The survey presented in this chapter focuses particularly on how mobility is modeled and on how mobile computing is supported.

In this thesis, mobility modeling, prediction, and proactive behavior are addressed. Thus, Section 3.1 gives an overview of how related methods are used by mobile and ubiquitous computing approaches. Section 3.2 details the mobility support provided by different types of middleware approaches including *grid computing systems* and *software agent systems*. Section 3.3 summarizes the insights of the survey and compares the different approaches in terms of mobility support.

3.1 Mobility Awareness, Prediction, and Proactivity

Recently, mobile and ubiquitous computing related research approaches have been proposed which provide means related to this thesis' contribution. These approaches do not explicitly focus on bridging the gap between mobility awareness and seamless distributed computing. However, the methods proposed and the results achieved are important in terms of:

- mobility modeling,
- extraction of movement or context properties including prediction, and
- proactive mechanisms.

3.1.1 Mobility Modeling

Mobility modeling is important for simulation of person or vehicle movement, for advanced user behavior modeling, and for prediction purposes, like, for example, predictor based cellular mobility management optimization [Cay02, dW03]. User movements are modeled either by stochastic processes, like proposed by the *Random Walk* or *Random Waypoint* mobility models, by applying topographical information, or by more realistic heuristic models. Stepanov et al. [Ste03] propose a meta-model based framework for mobility modeling used for simulation.

For extraction of user movement patterns from real-world traces, *(embedded) Markov models* or models originating from text compression (*LeZi-Update model*)

are commonly used [Cam02]. Ashbrook et al. [Ash03] describe a study on GPS based location trace extraction and user mobility modeling based on second order Markov models for locations. A first order Markov model is used by Song et al. [Son03] and compared to random mobility models in this work. Koukoutsidis et al. [Kou04] propose location prediction for arbitrary time intervals t in case the starting position ($t = 0$) is known by means of embedded Markov models. Here, prediction is used to optimize paging in mobile cellular networks. A model-independent information theory based approach using the Lezi-Update model for mobility and resource management in cellular networks is presented by Roy et al. [Roy04]. By interpreting the user's movements as a sequence of symbols (or a string), movement chunks are gradually built without prior knowledge.

Moon at al. [Moo03] describe a history based mobility predictor based on a particular algorithm developed to adapt QoS provisioning. Here, history trace tuples are matched against tuples in the database. The next mobility features assigned to the matching tuple are used for prediction purpose. Abdelsalam et al. [Abd04] focus on minimizing uncertainties of location tracking applications by relating context properties. *Bayes networks* are used to describe these causality relationships among a set of context values observed, like time, location, or additional environmental properties. Markoulidakis et al. [Mar97] distinguish three basic types of mobility models which describe user movement in an area, a street, and a street unit. The model consists of reasonable parameters which allow to describe user movement in a third-generation mobile telecommunications system based on transportation theory. Trips consisting of endpoints, purpose and routes, area zones characterized, for example, by population density and natural limits, and movement attraction points are some of the modeled characteristics.

3.1.2 Feature Extraction and Context Prediction

A variety of location estimation techniques may be used for generating physical traces by observation. For example, outdoor positioning can be based on the *Global Positioning System (GPS)* or GSM based methods which consider the *Time of Arrival* of signals or the *Angle of Arrival*. For indoor positioning systems, *Radio Frequency Identification (RFID)* or WLAN positioning, like the *Ekahau*[2] positioning engine, are reliable technologies.

Headon [Hea03] considers movement awareness based on sensing the ground reaction force in order to analyze and categorize user movements. Statistical pattern recognition is used for distinguishing between seven classes of movement, like crunching, jumping, sitting, or rising to stand. Applied to ubiquitous gaming, an *active floor* originating from Olivetti and Oracle Research [Add97] can be used as a natural interface for controlling movement of, for example, virtual

[2]http://www.ekahau.com/

avatars [Hea02]. Figure 3.1(a) depicts such a floor (titles are about 500x500mm). The *load cells* sense the vertical ground force of neighboring titles.

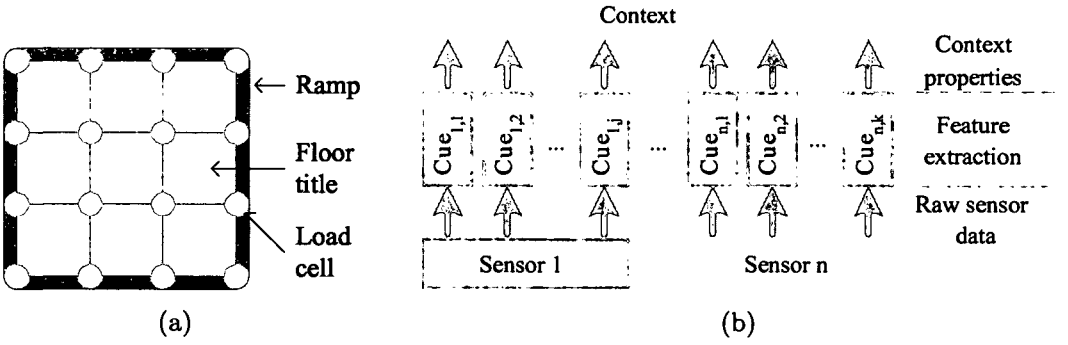


Figure 3.1: (a) Active Floor [Hea02] and (b) the multi-sensor extraction architecture overview [Gel02]

Embedded systems, like context-aware systems, usually rely on multi-sensor data. Layered architectures are used in order to sense, pre-process and aggregate, and analyze data as depicted by Figure 3.1(b). Gellersen et al. [Gel02] describe such a framework based on experiences from a variety of projects focusing on context-awareness. In a post-processing manner, *clustering* is applied to sensor data which has been recorded in different environmental situations.

Mayerhofer et al. [May03] discuss several clustering algorithms suitable to group multi-sensor data (see also the discussion in [vL01]). This work further presents promising results for features extracted from sensor data. For prediction purpose, a LeZi based algorithm has been implemented and Markov models are considered as well. Focusing on the coordination of a manifold of appliances generating events in a context-aware system, sensor data *time series analysis* is another method proposed for prediction of future context properties [Fer03].

PILGRIM is a location broker and mobility-aware recommendation system [Bru03a]. In contrast to the approaches discussed above, here, a persons' digital usage patterns are linked to locations and analyzed accordingly. Thus, spatial usage history patterns are extracted by observing locations visited and links used. Another advanced user centric work proposes next location prediction based on a *reasoning engine*. In this approach, the user's interest in places nearby, the user's schedule constraints, and the user's goals and tasks are considered [Sam03].

3.1.3 Proactive Mechanisms Addressed

Proactive context transfer based on Signal to Noise Ratio (SNR) differences between two WLAN access points is proposed by Duong et al. [Duo04]. Here, context transfer mechanisms are invoked whenever the difference in SNR values between two access points reaches a particular threshold. This mechanism allows to complete context transfer accurately before the handover takes place.

Salovaara et al. [Sal04] describe six modes of proactive resource management. Resources eventually needed in the near future have to be (i) *prepared for usage*, (ii) *optimized* in case of collaborative access, (iii) *recommended* to the user, (iv) *manipulated* in order to complete a proactive action like payment, (v) *inhibited* in case of risks, and (vi) *finalized*, that is, released, in case of near task completion.

3.2 Mobile Computing Middleware

Middleware is commonly defined as a software layer capable of masking heterogeneity of networks and operating systems. Additionally, this layer provides programming abstractions for distributed applications (see, for example, Coulouris et al. [Cou01], p. 32, or Tanenbaum et al. [Tan02], p. 37). Figure 3.2 depicts the general architecture.

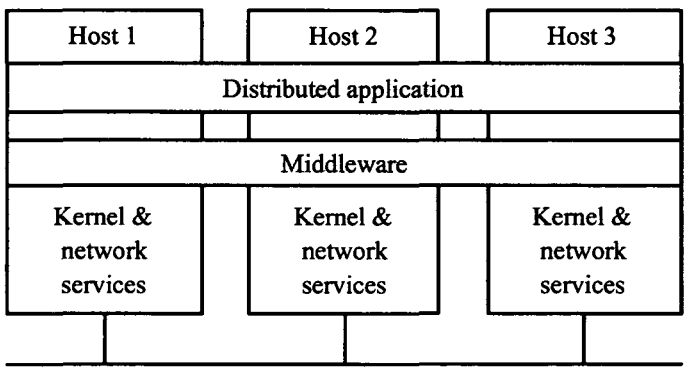


Figure 3.2: Distributed system architecture including a middleware layer [Tan02]

Although different types of middleware systems can be distinguished, no commonly accepted classification scheme exists. The structure of this survey is based on the detailed survey carried out by Mascolo et al. [Mas02a, Cap02, Mas04] and on major characteristics which are beneficial for mobile computing. The

assignment of characteristics and middleware types is given by Table 3.1.[3]

Characteristics	Middleware Types
Reusability and reconfigurability	Object based, component based
Asynchronous communication	Space based, data sharing oriented, asynchronous message passing based, (mobile) agent based
Awareness (context, QoS, application)	Context-aware, QoS-aware, application aware
Service discovery	Service discovery based, grid computing based
Reactivity	Reactive, event based

Table 3.1: Assignment of middleware types to mobility related characteristics

Reusability and Reconfigurability. Software reuse and reconfiguration is commonly addressed by *object-oriented* systems and languages. Similarly, *object based* or *component based* middleware approaches provide object centered communication support. These approaches naturally support mobile computing applications with means for reuse of objects and for object configurability.

Asynchronous Communication Support. Decoupling of dependent distributed processes can be achieved by means of asynchronous communication. In case of disconnections, asynchronous communication can still be provided. *Space based* approaches which originate from the Linda tuple space approach [Gel85, Gel92] implement asynchronous communication in a natural way. Additionally, *data sharing* middleware, *asynchronous message passing* middleware, and *mobile agent* based middleware provide means for asynchronous communication.

Awareness. Mobility caused changes are a major issue in mobile computing scenarios. Here, both changes in QoS parameters and application specific parameters are of specific interest. Accordingly, *context-aware* or *location-aware*, *application-aware*, and *QoS-aware* middleware approaches can be distinguished. While context-aware middleware systems[4] address adaptation based on sensing and interpreting the physical world, application-aware middleware propose middleware adaptations driven by application requirements. QoS-aware middleware systems focus on middleware adaptations due to changes of QoS parameters.

[3]Middleware approaches are assigned to types and characteristics in consideration of the authors' focus and self-classification.

[4]Here, location is seen as a part of the physical context.

Ease of Service Discovery. Since mobile hosts are expected to roam between different computing infrastructures, service registration and service discovery have to be provided. Middleware focusing on these issues is termed *service discovery based* middleware in this work. Since *grid* middleware focuses on resource sharing and management, they are referred to as approaches mainly addressing resource discovery and service discovery.

Reactivity. In addition to being aware of changes, middleware sometimes provides additional means to react to changes. Such approaches are termed *event based* middleware or, when focusing on means for reaction, *reflective* [Rom01, Cap03] or *reactive* middleware [Cab98, Cab00a].

Additionally, security is a very important issue for mobile computing approaches. Since this thesis does not address security issues, this aspect is omitted.

3.2.1 Object Based and Component Middleware

Remote Method Invocation (RMI)

Originating from *Remote Procedure Calling (RPC)* [Bir84], *Remote Method Invocation (RMI)* has been derived based on the object-oriented computer language paradigm. Instead of sending messages to and receiving messages from processes, the abstraction of method invocation allows to handle remote objects similarly to local objects. Furthermore, techniques like object serialization provide *parameter marshaling* functionality, that is, a translation of parameters into a machine-independent format. Hence, parameter exchange is supported between processes running on computer architectures using different character coding or number representations.

Java RMI[5] is a prominent example for RMI based on the Java programming language. Thus, Java RMI can be seen as lightweight middleware based on the definition used in this work. However, Java RMI does not provide any advanced middleware functions, like replication strategies, persistent data management or recovery policies.

Based on RPC, support for mobile clients is proposed by adding dynamic binding, disconnected operation and call retries in [Bak95, Bak96]. Modular *mobility support routers* act as proxies for clients connected over a wireless network. For Java RMI, Campadello et al. [Cam00] propose *performance enhancing proxies* in order to compensate the high protocol overhead of Java RMI. Here, mediators are used on the server node which implement some of the most costly operations.

[5] http://java.sun.com/products/jdk/rmi/

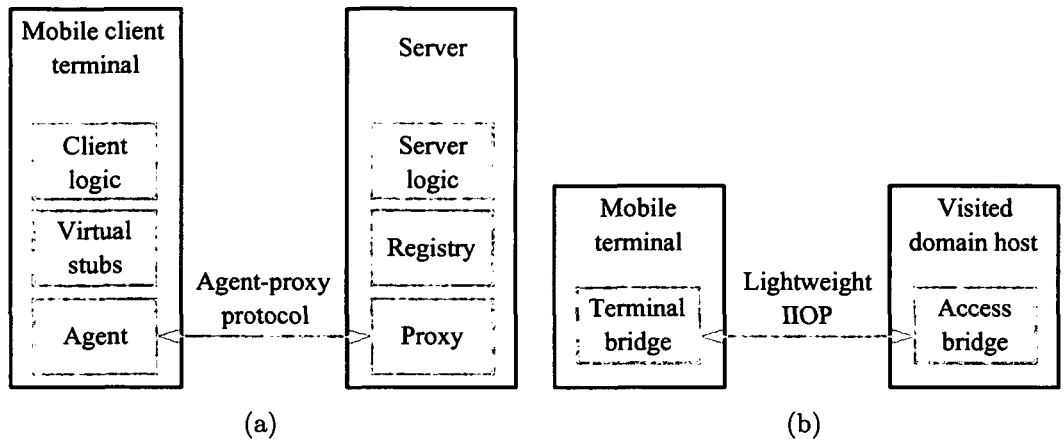


Figure 3.3: Communication overhead reduction (a) by Java RMI proxy extensions [Cam00] and (b) by bridges based on lightweight IIOP [OMG04b]

Enterprise Java Beans

Based on the Java programming language, *Enterprise Java Beans* can be used to develop Internet applications based on components (*beans*) [Sun03]. *Containers* provide middleware functionality, like persistence of data, transactions, and security. By means of, for example, *Java Server Pages (JSP)*, dynamic Web pages can be integrated into such a middleware.[6]

Object Request Brokers (ORBs)

The most prominent example of an object request broker is the *Common Object Request Broker Architecture (CORBA)* defined by the *Object Management Group (OMG).*[7] CORBA provides interoperability, in particular between different programming languages. The core module of this middleware is the *Object Request Broker*, which enables communication between distributed processes by providing links to required objects. Methods can be invoked and the *callback mechanism* can be used for result retrieval by clients which have requested remote services. The *Internet Inter-ORB Protocol (IIOP)* which implements the framework *General Inter-ORB Protocol (GIOP)*, provides interoperability between different ORBs [OMG04a].

Based on the results achieved by the *DOLMEN* project [Lil97], the OMG enhanced CORBA for wireless networks by means of two *bridges*, which establish

[6] http://java.sun.com/j2ee/
[7] http://www.omg.org/

a lightweight IIOP communication tunnel between a mobile terminal and a visited domain. Figure 3.3(b) depicts this mechanism. The *access bridge* furthermore acts as a proxy. Thus, the total message flow can be reduced.

Distributed Object Based Approaches

For developing wide-area distributed applications, *Globe* has been proposed [Bak00, vS99]. Globe's object model is based on (physical) replication of objects as depicted by Figure 3.4(a). Each object encapsulates state and decomposition information. The object can be decomposed into several sub-objects, like a communication sub-object, a replication sub-object, or a control sub-object. The replication mechanism can be exploited by mobile terminals which require access to such a distributed object. Terminals may choose the host of a DSO based on best network conditions. Additionally, Globe is able to integrate Web-documents as *semantic sub-objects.*[8] *AspectIX* is a CORBA compliant middleware which adopts the Globe fragmented object model. Here, replication is used for step-wise migration of objects [Gei98, Hau01].

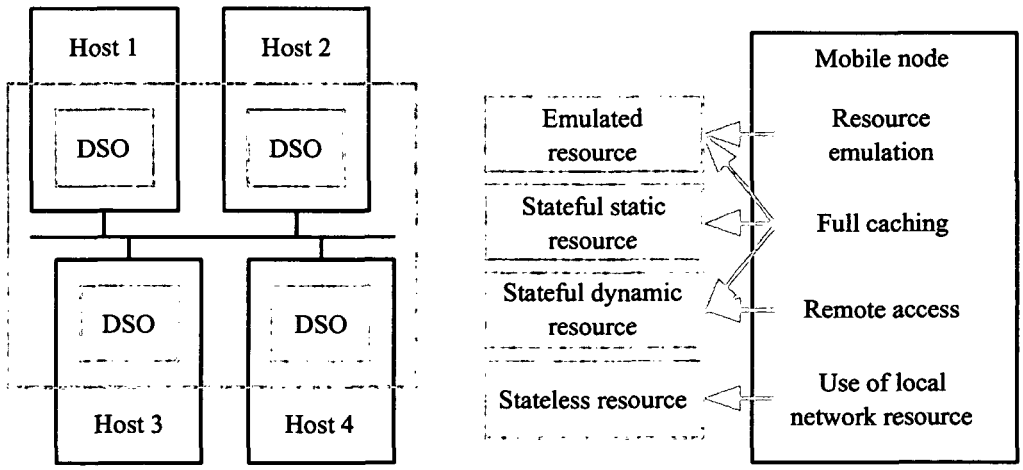


Figure 3.4: (a) Globe Distributed Shared Objects (DSOs) [Tan02] and (b) resource based operation selection [Sch95]

Distributed Computing Environment (DCE)

DCE is still an important industry standard which offers *remote procedure calling*, *directory services* for naming, *security services* for access and authentication

[8] A detailed description of Globe is given by Tanenbaum et al. [Tan02], pp. 545–565.

purposes.[9] Schill et al. [Sch95] describe a DCE architecture for mobile resource management based on *caching* and *message queuing*. Similar to the approach presented by this thesis, connection states are modeled and resource access behavior is changed according to these states. Therefore, a detailed resource classification is introduced. Resources may either be *stateful* or *stateless*. In the latter case, any other type of resource in the local environment may be used instead. Stateful resources are further distinguished into shared (*dynamic*) resources, resources accessed only by one single process (*static*), and resources which can be emulated (*emulated*). Figure 3.4(b) depicts the operations possible during degraded link quality and disconnection times (gray boxes) depending on the type of resource. Microsoft's *Distributed Component Object Model (DCOM)*[10] uses DCE RPC.

3.2.2 Space Based Middleware

Space based middleware provides a communication infrastructure by means of a globally available persistent *virtual shared memory*. Based on the *Linda* programming primitives [Gel85, Gel92], remote processes interact via writing to the space and reading from the space in either a consuming or a non-consuming manner. Code migration is enabled by *spawning* processes at remote sites. Additionally, recent space based middleware approaches support the notification of space changes by means of events. Angerer surveys different space based middleware systems in [Ang02].

Most of space based middleware approaches are based on the *Linda Tuple Space (TS)* model [Gel85, Gel92]. Here, data tuples, that are, sets of ordered typed fields which might be specified or unspecified, are used to write data to the space and *template tuples* are used to retrieve appropriate tuples by simple matching algorithms. Shared data (or shared objects) can be accessed directly by means of *Object IDentifiers (OIDs)*. In contrast to tuples, OIDs support creating linked data structures. Such approaches are termed *Shared Object Space (SOB)* approaches [Küh02]. Figure 3.5(b) depicts the differences between TS and SOB approaches.

Tuple Spaces (TS)

TS based middleware approaches are widely seen as promising approaches for ubiquitous and mobile environments. For example, the *Stanford Interactive Room (iRoom)* [Bor02] uses an extended tuple space model for collaboration support in a ubiquitous computing workspace by means of an *event heap* for event processing [Joh03].

[9] http://www.opengroup.org/dce/
[10] http://www.microsoft.com/com/

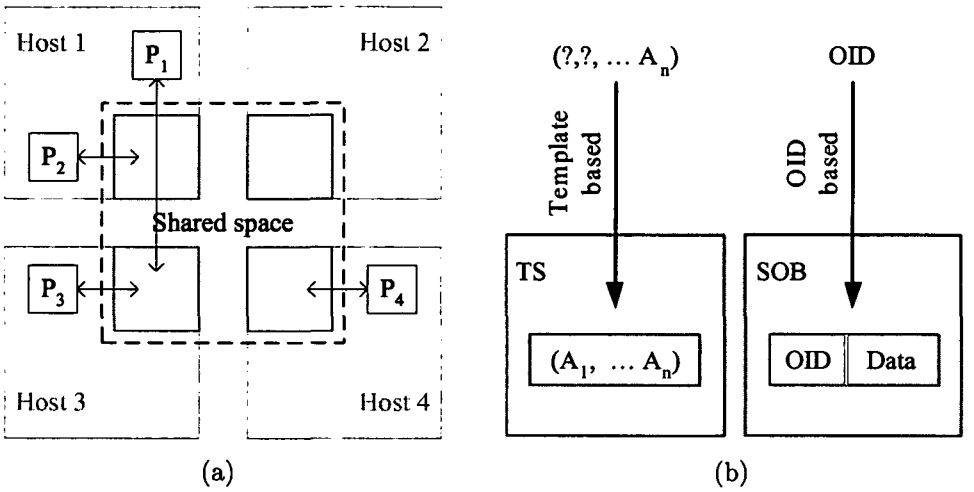


Figure 3.5: The (a) shared DS paradigm and (b) two different sub-categories: TS and SOB

Focusing on the support of mobile components, *Linda in a Mobile Environment (LIME)* introduces the notion of an *Interface Tuple Space (ITS)*. The ITS defines a *mobile agent's (MA's)* local TS (or agent context) which is mapped to a hierarchically structured TS, whenever the MA enters a logical area [Mur01, Pic00]. The in() and out() operations from the basic Linda model are altered by adding a *location address variable* for the tuple's destination, which allows the tuple to migrate to a host as soon as the destination is available. Furthermore, a notion for reaction is used to change the TS. The semantics of these reactions are based on *Mobile UNITY* [Rom02]. Figure 3.6(a) depicts the approach. Here, MA 3 roams to a new logical area which causes the corresponding ITS to migrate as well. Thus, the topology of this distributed shared TS is changing dynamically caused by agent and device migration.

Tuples On The Air (TOTA) proposes a TS based context-aware middleware, which allows to specify the management of tuples on application level [Mam03]. Here, tuples are spatially distributed on mobile devices and propagated across the network based on application-specific rules. Mobile computing is supported by means of replicated tuples in L^2imbo [Dav98]. Consistency between the distributed TS sites is provided by daemon processes. Hence, in case a host disconnects, any replica can be used instead.

JavaSpaces, which is included in the *Jini* framework [Bis03, Edw99], and *T-Spaces* [Leh99] are further widely referred TS approaches based on associative search for data tuples. However, these approaches do not explicitly provide specific support for mobile devices.

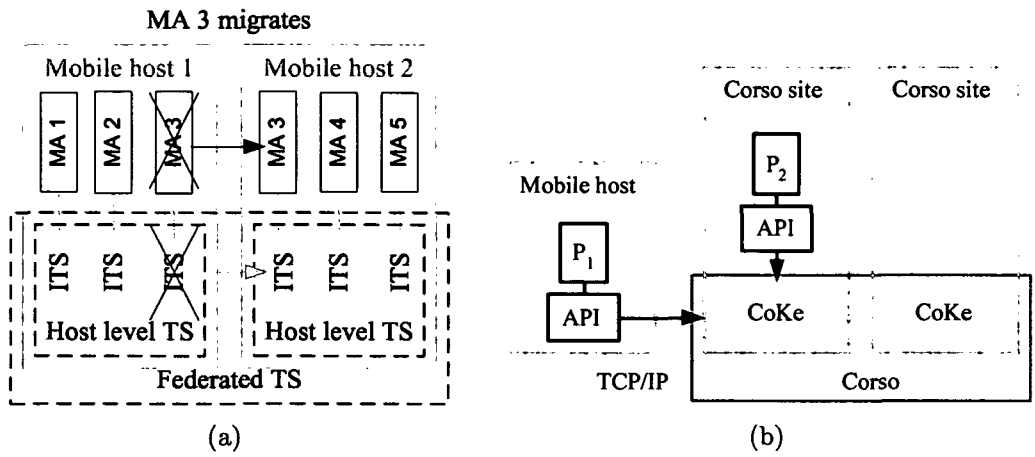


Figure 3.6: (a) Linda in a Mobile Environment (LIME) [Pic00] and (b) CORSO [Küh01, Küh02]

Shared Object Based Spaces

In contrast to associative search in a tuple space, the distributed *CORSO* implementation provides object ID based access to data items and advanced means for reliable replication, caching, garbage collection, and transactional processing [Küh98b, Küh98a, Küh01, Küh02]. Furthermore, *Application Programming Interfaces (APIs)* are provided for a variety of programming languages, like Java&Co and C++&Co. In case a mobile device, like a PDA or a smartphone, is not able to run a full CORSO site locally, this device may use the Java&Co API library to connect to a remote site and thus access the shared object space as depicted by Figure 3.6(b). A detailed introduction to CORSO is given in Section 6.1.

3.2.3 Data Sharing Oriented Middleware

XMIDDLE is a data-centric middleware which enables sharing of XML documents among mobile hosts [Mas02b, Zac02a]. By using XML as document exchange format, heterogeneous hosts and applications can be integrated. In order to provide access in disconnection states, XMIDDLE proposes the use of replicated documents. Hence, XMIDDLE addresses copy, synchronization, data locking, and release of data locks in a mobile environment. Versioning is used in order to solve inconsistencies while synchronizing. Similar to the approach presented in this thesis, XMIDDLE can only facilitate working on copies, in case the disconnection procedure has been finished successfully. However, in contrast to the approach proposed in this thesis, no means for accurate or proactive detection

of possible disconnections are provided.

In the *Bayou* replicated storage system for mobile environments, consistency is achieved by client side per-write conflict resolution. On the server side, *rollback* and similar write operation serialization assures eventual consistency [Ter95].

3.2.4 Context-Aware Middleware

In ubiquitous computing environments, context-awareness is one of the main focus research areas. In the *CoolTown* project, users are envisioned as nomads, as described by Kindberg et al. [Kin01]. Interactions are assumed to be spontaneous and context-dependent. From a collaborative computing perspective, a *Pervasive Information Community Organization (PICO)* supports mission oriented communities by using two abstract basic building blocks: (i) *delegents*, which represent, for example, users or applications, and (ii) *camileuns*, which represent devices. Methods for acquiring and disseminating information include context-aware services and caching, prefetching, and push-caching policies [Kum03].

The *Mobile Collaboration Architecture (MoCA)* is another middleware focusing on context-aware collaboration for mobile users [Sac04]. In addition to modular context sensing services, MoCa uses *proxy technology* in order to reduce network load and load balancing. Takasugi et al. [Tak03] propose seamless service migration following the user's movement by means of *seamless proxies* which manage a particular area.

In contrast to many other approaches which add context-aware services to existing middleware approaches, the *Nexus* middleware architecture has been designed for context-awareness. A three tier architecture is proposed, consisting of (i) a *service tier*, where context-servers are situated, (ii) a *federation tier* which consists of components responsible for service discovery, and (iii) a *application tier* where applications reside (see Figure 3.7(a)). The federation tier's components communicate with the other layers by means of *event notifications* and *request/answer* communication style. Distributed Nexus nodes communicate by means of the *Simple Object Access Protocol (SOAP)*.[11] Nexus provides a hierarchical *context world model* based on the concept of inheritance. Thus, services particularly fitting ubiquitous computing scenarios can easily be introduced, like a *geocast* service used for multicasting in a specific geographical area [Dür04, Fri02, Nic04].

Context is modeled and introduced to existing coordination concepts in various ways. Hawick et al. [Haw03] extend the concept of user preferences with spatial or temporal information and introduce the new term *active preferences.* In some approaches, context is modeled in terms of objects, like *location aware*

[11] http://www.w3.org/TR/soap/

remote objects [Jär04] and *sentient objects* [Sor04]. In the latter case, *publish/subscribe* communication is used for context change notification.

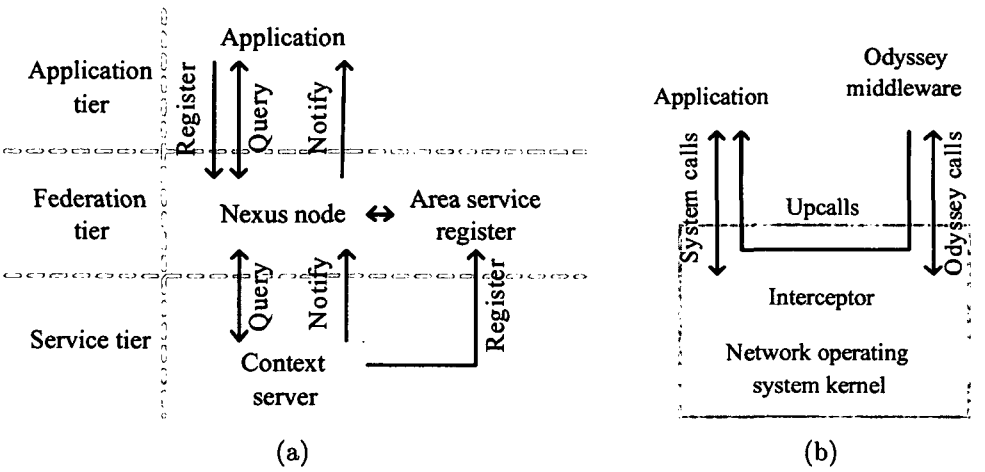


Figure 3.7: Concept of (a) the context aware middleware Nexus [Dür04] and (b) application-aware computing proposed by Odyssey [Nob97]

The *Rover* toolkit supports both *mobile-transparent* and *mobile-aware* computing [Jos95] including mechanisms, like priority based scheduling of messages, computation relocation, that is, migration of code, notification of changes in the environment, and object replication including consistency management. Similar to CORSO, Rover uses primary-copy based concurrent access. Another appealing context-aware approach is presented by Conan et al. [Con04]. Here, a *connectivity manager* monitors the availability of network resources by means of detectors which are assigned to each resource. These detectors use hysteresis mechanisms in order to smooth variations and to detect the connectivity mode in a stable manner. Depending on the connectivity mode, the sending of requests is altered.

From a computer language perspective, Popovici et al. [Pop03] use define proactive activities for embedded applications by language primitives. Based on *Aspect Oriented Programming* [Kin97], proactive operations are introduced by the language primitive *before*. *Context UNITY* [Jul04] is another language based approach for context modeling based on *Mobile UNITY* [Rom02]. In addition to defining places for code execution explicitly, other context characteristics can be defined in the application.

Context-awareness is further addressed by *reflective* and by *event based* middleware approaches.

3.2.5 Application-Aware Middleware

Application-awareness addresses the interaction between application, middleware, and (network) operating system. Consequently, the middleware layer does not hide all network properties. Peddemors et al. [Ped04] propose an approach which provides information about lower network and mobility management protocols to mobile host applications. For example, the release of a modem connection may be forced by the application in case good WLAN connection is detected.

Application-awareness is also implemented by *Odyssey* on top of the *NetBSD Kernel* [Nob97]. Applications can operate on the middleware's objects and express resource requirements, as well as receive events in case the resource requirements are no longer met. Operations on Odyssey objects are redirected by means of an *Interceptor*. Figure 3.7(b) depicts a simplified version of the Odyssey architecture.

The *Kernel-Middleware eXchange (KMX)* is a cross layer approach which allows not only horizontal cooperation among hosts, but also vertical cooperation between the layers of nodes [Sam04].

The *Gaia Meta-Operating System* and application framework enables to support *active spaces*, that are, digitally augmented physical spaces [Rom00b, Rom02, Rom03]. Thus, the framework consists of several components supporting the integration of a variety of devices and services. From a middleware perspective, Gaia is classified as a component based middleware platform. The Gaia kernel consists of a *component management core* and a set of additional services, like a *context service*, a *presence service*, and an *event manager service*.

3.2.6 QoS-Aware Middleware

QoS-aware middleware approaches for mobile computing provide means for adaptations based on QoS changes. Device characteristics observed are battery life time or automatic invocation of the sleeping mode during idle times. Wireless network characteristics monitored are based on common network performance measures, including throughput, latency, and signal strength. Here, the *effects of movement* can be observed [Cha99].

Modular architectures, like proposed by Menasce [Men04], encapsulate QoS in *QoS-aware components* responsible for QoS monitoring. These QoS-aware components can be integrated in applications and, thus, make applications QoS-aware.

Nahrstedt et al. [Nah01] propose an architecture for ubiquitous computing environments based on *QoS proxies* residing on mobile clients and on application servers, which allow to adapt to QoS changes, to freeze the current resource allocation and QoS configuration state by means of *snapshots*, and to (re)allocate

resources at new destinations. Both user mobility and terminal mobility are supported by this approach.

For mobile multimedia communication, *Mobiware* implements mature strategies for wireless ATM networks based on CORBA technology [Cam97]. This QoS-aware middleware aims at hiding handoff periods and periods of persistent QoS fluctuation from the application. *Active Transport Objects (ATO)* implement the major strategies, that are: (i) *mobile filters*, which allow to drop or compress specific data streams, for example, video connections can be dropped before dropping audio channels, (ii) *mobile error control modules*, which may add additional reliability to protocols in times of high network error frequency, and (iii) *mobile snooping modules*, which implement snooping of end-to-end protocols.

3.2.7 Middleware Focusing on Service Discovery

In mobile environments, (i) *service discovery*, (ii) *service registration and advertisement*, and (iii) *feature or driver downloads* are major issues. Hence, a group of frameworks for ubiquitous and mobile environments focus particularly on these challenges.

The goal of Sun's *Jini*[12] is to federate users and resources needed. The key concept in the Jini architecture is a *service* [Sun99, Edw99]. Services can be registered, joined, and discovered. Communication between services is achieved by means of Java/RMI. In Jini, service access is based on leases, which define the time periods granted for service access. An appropriate lease period has to be chosen in order to make trade-offs between resource utilization, responsiveness, and system size. Bowers et al. [Bow03] propose self-adaptive algorithms for better lease variation. From a coordination perspective, Jini integrates the tuple space implementation *JavaSpaces* [Sun00, Bis03], which supports four operations: (i) write, (ii) read, (iii) take, which is a consuming read, and (iv) notify, which can be used for object notification.

Universal Plug and Play (UPnP)[13] provides a framework for services and devices. By means of standard protocols, like SOAP or HTTP, distributed components interact. Service discovery is realized by means of the *Simple Service Discovery Protocol (SSDP)* which allows services to gracefully leave the network [Mic00]. In contrast to Jini, which is related to Java, or UPnP which is related to Microsoft systems, *Salutation*[14] aims at providing a platform independent solution [Pas99].

Lindemann et al. [Lin03] exploit epidemic data dissemination in order to support lookup while devices are weakly connected or disconnected from an infras-

[12] http://java.sun.com/products/jini/
[13] http://www.upnp.org/
[14] http://www.salutation.org/

tructure network. *Passive Distributed Indexing (PDI)*, that is, caching of registry information by mobile devices, is used in such situations.

3.2.8 Event Based Middleware

In *publish/subscribe* or *event based* middleware approaches, processes communicate via generating and consuming events. In contrast to the synchronous *Publish/Subscribe* coordination pattern described in Section 4.3, here asynchronous event notification is assumed.

Figure 3.8(a) depicts the logical architecture of *JEDI* which is a typical publish/subscribe middleware [Cug01]. Events are generated by *active objects* (also termed *event sources*) distributed by *event dispatchers* (also termed *event brokers*) and received by *active objects* (also termed *event destinations*). Event distribution can either be local, topologically or hierarchical structured, or performed in an unstructured manner. The event dispatchers can be organized in a centralized manner, distributed, or replicated as detailed by Huang et al. [Hua04].

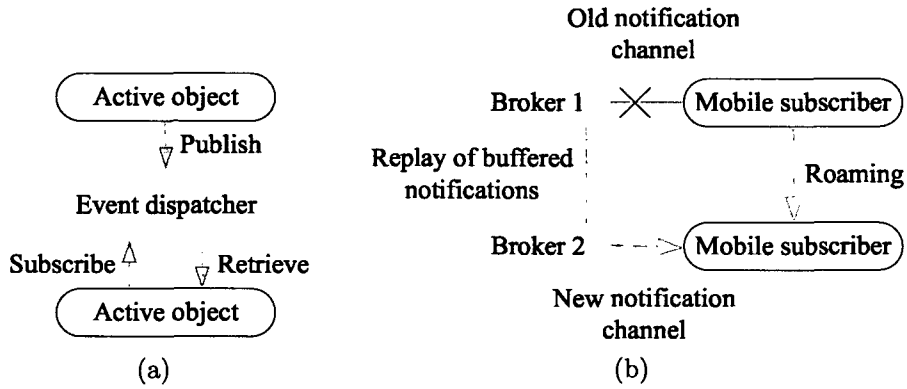


Figure 3.8: Concept of (a) the JEDI [Cug01] publish/subscribe logical architecture and (b) replaying of notifications for a roaming client as supported by Rebeca [Müh04]

Mobile location-aware clients produce significant network load by sending location updates periodically to brokers. This network load can be reduced significantly by solving spatial matching directly at the mobile device [Che03a]. In case mobile clients disconnect, buffering is commonly used to store the events intermediately at broker components [Tar03]. By means of migrating *modulators*, mobile clients are able to subscribe efficiently to new brokers as proposed by Chen et al. [Che03b]. However, Farooq et al. [Far04] state that handoff protocols

between two brokers are not efficient from a performance perspective in case of highly dynamic mobile clients.

Rebeca is a publish/subscribe middleware which provides advanced means for mobility support [Fie03, Müh04]. Buffering of notifications in combination with replaying these buffered notifications by a new supporting broker enables seamless event forwarding for roaming clients as depicted by Figure 3.8(b). Additionally, past notifications can be accessed by means of histories, notification delivery rates can be adjusted to the capabilities of the mobile device, and events can be filtered.

In addition to buffering (or logging), Burcea et al. [Bur04] introduce prefetching of notifications based on *mobility patterns* similar to the approach presented in this thesis. However, mobility patterns are in this approach random patterns based on randomly chosen directions. Similarly, Cilia et al. [Cil03] propose pre-subscription based on prediction of places that might be entered in the near future that is given by a *movement graph.* Replaying notifications allows mobile clients to re-synchronize with the current system state by *looking into the past.*

3.2.9 Reflective Middleware

Reflection, that is, the ability of a system to reason about and alter its own behavior, is a powerful means to enable middleware inspection and run-time reconfigurability depending on changes in terms of context, network resources, and computing power. Meta-space models are used in order to alter middleware behavior. Middleware providing reflection is sometimes termed *next generation middleware* because adaptability and flexibility required in ubiquitous and mobile computing environments is addressed right from the beginning [Eli99, Rom01, Cap02]. From a language perspective, Busi et al. [Bus02] define new primitives for the Linda tuple space calculus, that are, a *monitor* primitive, which spawns a listener process basically waiting for state changes and a process that notifies about state changes.

Object Request Brokers (ORBs) enhanced by reflection use meta level components in order to trigger the adaptation of base level middleware support. *OpenORB* proposes a meta-space model which distinguishes between the internal architecture view and the interface view [Bla01]. *Dynamic TAO* is a CORBA compliant dynamically configurable middleware [Kon00] which achieves reflection by exporting an interface for (i) transferring components, (ii) loading modules into the ORB during runtime, and (iii) modifying the ORB's configuration state. *OpenCorba* is another reflective CORBA implementation which adapts the behavior of the broker at run-time by means of explicit meta-classes [Led99].

CARISMA is a context-aware reflective middleware which aims at supporting mobile applications. In contrast to traditional distributed object and space based approaches, CARISMA opens the middleware to be programmable by applica-

tions using reflection [Cap02, Cap03]. The drawbacks of this approach are the loss of transparency and the occurrence of conflicts whenever applications require different middleware behavior at the same time. However, applications benefit from adapting middleware behavior based on application specific information, for example in terms of performance or reliability. Figure 3.9(a) depicts the approach. Aspects of middleware internals are made explicit using *meta-information*, while applications can alter middleware internals using a *meta-interface*. Similarly, *mChaRM* uses reification of communication channels [Caz02]. Communication channels are modeled as objects and thus, reflection about communication channels is based on meta-objects.

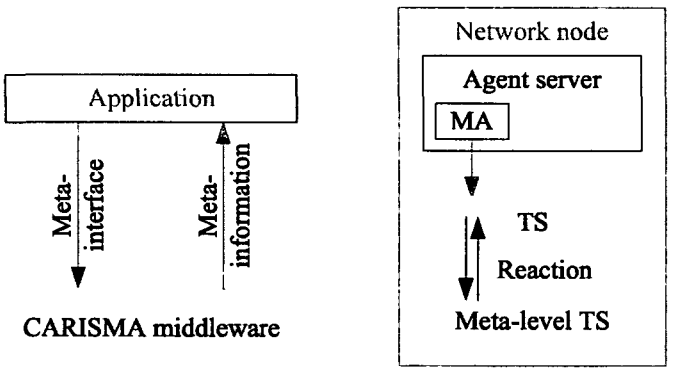


Figure 3.9: Concept of reflection proposed (a) by CARISMA [Cap03] and (b) by MARS [Cab98, Cab00a]

MARS [Cab98, Cab00a] is a reactive tuple space (TS) implementation, which allows to invoke actions upon events as depicted by Figure 3.9(b). Actions are used to change the tuple spaces' content and provide logical and space context information to mobile agents (MAs). Cabri et al. [Cab03] demonstrate, how MARS can be used to deploy location-dependent services for mobile users. In order to support mobile agents on mobile devices with physical context information, the TOTA middleware has been proposed [Mam03].

3.2.10 Message-Oriented Middleware

Although synchronous message passing is counterproductive in mobile distributed computing systems, asynchronous message passing can be of interest for mobile computing. For example, the *Java Message Service (JMS)* provides *peer-to-peer* messaging by means of queues which decouple processes to a certain degree and *publish/subscribe* communication style [Sun02]. JMS can be integrated with Java

Enterprise Beans. Musolesi et al. [Mus04a, Mus04b] describe adaptation of JMS for mobile ad-hoc environments. This approach is based on an *Epidemic Messaging Middleware for Ad hoc networks (EMMA)*, that is, messages are replicated on all connected hosts following the metaphor of an epidemic. Thus, in case a mobile host moves, this host can both force the *infection*, that is, the dissemination of events, and retrieve a message which has already arrived at the new network destination.

WebSphere MQ (Message Queue) (formerly, MQ Series) is another middleware based on the concept of distributed queues [IBM03]. Queue managers are responsible for receiving messages on behalf of processes. Channels may be opened between remote queue managers and multi-hopping is supported in case to remote queuer managers are not directly connected. Vivek et al. [Viv03] describe how to use these message queues for adding a messaging backbone to WWW distributed link services.

3.2.11 Grid Middleware

Grid technologies originate from the idea to provide computational power similarly to power grids. Instead of investing into more and more powerful supercomputers, the grid paradigm envisions the usage of the existing variety of heterogeneous computing resources. From the user's perspective, terminals of any type should be turned into supercomputers by connecting to the *computational grid*. In 2001, the European *DataGrid* project[15] led by CERN (European organization for nuclear research) started and is currently a well-known operating grid for analysis of huge amounts of scientific data. Additionally, Angerer et al. [Ang04] argue that grid computing is going mainstream by including more application areas than scientific computing and by fostering the convergence of Web-services and Grid computing. One of the most prominent approaches is the *Globus* meta-computing infrastructure toolkit.[16] Globus aims at providing an abstraction from the distributed nature of the computing components by defining services for resource and data management, for communication support, and for authentication purposes [Fos97]. The structure of Globus is based on a set of toolkit modules, like the Nexus communication module [Fos96] which provides abstractions from messaging like remote service requests, the *Metacomputing Directory Service (MDS)* [Las97], and a grid manager which is responsible for job assignment. Relying on the Globus toolkit, the *NorduGrid* middleware has been developed for high energy physicists.[17] High-level services have been re-written or created and a specific information schema has been introduced to the MDS [Smi03].

[15] http://eu-datagrid.web.cern.ch/eu-datagrid/
[16] http://www.globus.org/
[17] http://www.nordugrid.org/

Unicore is another meta-computing architecture which aims at providing a uniform interface for job preparation and a management of resources covered by the system. Abstract task objects containing information about resources needed are translated into batch jobs on the host selected [Rom99]. Recently, Unicore has been adapted an can be used on top of Globus.

Legion[18] is a grid middleware approach based on object orientation. The successor of Legion is the commercial system *Avaki*[19] focusing now on data grids. Objects are used to manage resource access by exporting a set of mandatory member functions, like state saving, requesting objects, and restoring of states. A grid portal enables access via commonly used Web technology [Lew96, Nat02].

Grid middleware approaches generally provide means to include heterogeneous computers into a computational and data grid. However, mobile devices are not specifically addressed by the approaches mentioned above. Here, the limiting capabilities of mobile devices prohibit the use of common APIs and network connection changes cause additional management overhead for both mobile grid clients and mobile grid resources. Recently, some grid approaches have been proposed addressing mobile computing termed *mobile grids* or *wireless grids*. Hwang et al. [Hwa04] propose a layer, which integrates mobile devices to existing grid platforms by means of *proxy processes*. Thus, mobile devices which are not capable of implementing the grid API can both register their services and request grid based computation by means of message passing with a proxy.

Gonzales-Castano et al. [GC02] propose a hierarchical design methodology for grid access from lightweight mobile devices and show the feasibility of this approach by means of prototypical implementations based on the *Condor*[20] grid middleware. Mobile clients access the grid middleware via Web technologies, like HTTP and WAP.

Bruneo et al. [Bru03b] investigate three communication paradigms for mobile (data) grid users, that are, *mobile agents*, *remote evaluation*, and *client server* by means of experimental and analytical simulation with varying available bandwidth. The results indicate that different paradigms should be used depending on the available bandwidth. For example, in case of low bandwidth, mobile agents achieved the best results. Future work of the authors addresses the implementation of an extension to Globus for mobile users based a mobile agent platform.

Kurkovsky et al. [Kur03, Kur04] propose a centralized problem solving environment for mobile devices based on a computational grid. Brokering and an advanced heartbeat algorithm (termed *keep-alive protocol*) is used to manage mobile computing nodes based on multi-agents.

[18] http://legion.viginia.edu/
[19] http://www.avaki.com/
[20] http://www.cs.wisc.edu/condor/

Agent Based Middleware

In contrast to physical mobility, logical mobility addresses migration in the digital space. Mobile code which decides autonomously upon migration is termed *mobile agent* [Whi97, Lan98, Pic97]. Mobile agents have gained popularity in the recent decade and are described as an alternative communication style compared to client/server communication or peer-to-peer computing [Cou01], p. 38. Benefits of mobile agents are (i) encapsulation, (ii) reduced bandwidth consumption, and (iii) support for disconnected operations [Mur01].

The *Secure and Open Mobile Agent (SOMA* framework for distributed programming is a Java based platform for mobile agents [Bel00, Bel01]. Mobile agents support disconnected resources by caching messages and by delivering messages upon reconnection. SOMA is a modular framework which supports naming by means of globally unique identifiers, user mobility by establishing a virtual environment for the user, independent of his or her location or the device used, and mobile virtual terminals which support terminal mobility. Additionally, virtual resource management on the server side allows to establish connections between resources and mobile clients dynamically.

In more recent work, Bellavista et al. [Bel03a] describe how mobile terminals can be supported by proxy mobile agents. The context-aware and, in particular, location-aware middleware *SCaLaDE (Services with Context awareness and Location awareness for Data Environments)* implements mobile agents which follow mobile terminals on the move. Additionally, in [Bel04] the authors describe how such a mobile agent migration can be optimized by means of user mobility prediction based on *Grey* discrete models which allow to describe uncertain memberships [Fan04].

The *Java Agent DEvelopment (JADE)* [Bel99] framework is an agent based software platform compliant to FIPA specifications.[21] Messaging between agents is based on Java RMI. For mobile devices, the *Lightweigth AgEnt Platform (LEAP)* based on JADE is proposed by Bergenti et al. [Ber01]. By proposing another lightweight FIPA compliant agent platform, Hattori et al. [Hat03] achieve major advances by reducing the program size at the mobile device.

3.3 Discussion of Middleware Approaches

In order to compare the middleware approaches discussed in Section 3.2, the most promising representatives of each category are summarized by Table 3.2. Particularly, the selection considers approaches exhibiting methods similar to the contribution of this thesis (highlighted by an asterisk (*) in the table). The

[21] http://www.fipa.org/

following mobility related characteristics are compared:

Type. The middleware type corresponds to the classification used in the previous section. Middleware types are distinguished as follows: (i) *OB (Object Based)*, (ii) *CB (Component Based)*, (iii) *TSB (Tuple Space Based)*, (iv) *SOB (Shared Object Based)*, (v) *DSOO (Distributed Shared Object Oriented)*, (vi) *CA (Context-Aware)*, (vii) *AA (Application-Aware)*, (viii) *QA (QoS-Aware)*, (ix) *SDB (Service Discovery Based)*, (x) *EB (Event Based)*, (xi) *R (Reactive)*, (xii) *AMB (Asynchronous Messaging Based)*, (xiii) *GB (Grid Based)*, and (xiv) *AB (Agent Based)*.

Paradigm. Here, the communication paradigm is addressed. The following types are used: (i) *(MP) Message Passing*, (ii) *VSM (Virtual Shared Memory)* based, (iii) *RMI (Remote Method Invocation)*[22], (iv) *ROA (Remote Object Access)*, like provided by SOAP, and (v) *MA (Mobile Agent)*. Furthermore, it is explicitly stated by a (+) in case exchanged data is stored persistently by the middleware.

m-Notion. In case the approaches present a notion for user, terminal, or logical mobility, this notion is presented in a descriptive manner.

Awareness. This characteristic is used to compare the physical context properties addressed by the middleware approach in a descriptive manner.

m-Devices. Since mobile devices exhibit limited footprints, adaptations dedicated to lightweight implementations are investigated and described.

m-FT. Mobility caused failures require additional fault tolerance mechanisms. Accordingly, these middleware mechanisms are listed.

SD (Service Discovery). If applicable, service registration, lookup, and advertisement mechanisms are described here.

m-SW. Middleware approaches are investigated due to support for code mobility. The following types are distinguished: (i) *MA (Mobile Agent)*, (ii) *MC (Mobile Code)*, and (iii) *PS (Process Spawning)*.

Among the middleware approaches presented, a few address mobile coordination challenges similar to the contribution of this thesis. The work on DCE based resource sharing [Sch95] addresses caching, local usage of resource, and emulation of resources. Similarly, XMIDDLE [Mas02b] proposes the use of replicated data in case of disconnections. Both approaches offer reactive mechanisms which provide no explicit means for accurate completion of copying operations.

[22]For simplicity reasons, *Remote Procedure Calling (RPC)* is assigned to this category.

Proactive behavior based on movement prediction is addressed by the recent works of Burcea et al. [Bur04], Cilia et al. [Cil03], and *ScAlADe* [Bel04]. The first two works use heuristic mobility predictors for event dissemination, while ScAlADe uses grey theory based mechanisms to model the uncertainties of prediction for proactive mobile agent migration.

To summarize and to conclude, it has been argued that mobility caused disconnection times or times of weak wireless link connectivity can be compensated by working on copies and replication. However, since in particular the process of creating copies needs to be invoked before the network conditions are too bad, reactive behavior is not suitable to overcome this problem. The approach proposed by this thesis aims at proactive and accurate invocation of compensation activities by means of predictors based on mobility models. First promising results of this approach have been discussed for the producer/consumer coordination pattern in [Hum04a]. This thesis enhances these results by applying the approach to more reference coordination patterns which allows to investigate the potentials and limits of this approach.

Approach	Type	Paradigm	m-Notion	Awareness	m-Devices	m-FT	SD	m-SW
RMI-proxies [Cam00]	OB	RMI	-	-	proxy	proxy	registry	-
wireless CORBA [OMG04b]	OB	RMI	-	-	lightweight IIOP	proxy	registry	-
DCE m-resource [Sch95]*	CB	RMI+	-	connection states	-	caching, local emulation	registry	-
LIME [Mur01]	TSB	VSM+	-	location variable	-	MA, reaction	-	MA
L²imbo [Dav98]	TSB	VSM+	-	-	-	replication	-	MA
CORSO [Küh01, TEC04]	SOB	VSM+	-	-	m-CORSO	replication recovery	-	PS
XMIDDLE [Mas02b]*	DSOO	MP+		connection state	-	replication versioning	-	-
Nexus [Dür04]	CA	ROA	-	context world model	-	-	federation tier	-
Rover [Jos95]	CA	MP	mobility awareness	notification	-	replication, MC, scheduling	-	MC
Odyssey [Nob97]	AA	MP	-	notification	-	-	-	-
Mobiware [Cam97]	QA, OB	RMI	-	network QoS	-	filters, snoop, error control	-	-
Jini [Edw99]	SDB	VSM+	-	-	-	lease	register, join, discover	PS
JEDI [Cug01]	EB	EP	-	-	-	distributed EP	notification	-
Burcea et al. [Bur04]*	EB	EP	m-pattern	notification	-	pre-fetching	notification	-
Cilia et al. [Cil03]*	EB	EP	movement graph	notification	-	pre-subscription	notification	-
CARISMA [Cap03]	R, AA	ROA	-	application	-	application	-	-
MARS [Cab00a]	R, TSB	VSM+	-	meta TS	-	meta TS	-	MA
JMS [Sun02]	AMB	MP+	-	-	-	-	-	-
EMMA [Mus04b]	AMB	MP+	-	-	-	epidemic replication	-	-
Kurkovsky et al. [Kur04]	GB	MP	-	network QoS	PDA int.	keep-alive	broker	-
SOMA/ScAlADe [Bel04]*	AB	MP, MA	grey based	location	- -	proactive MA proxy	-	MA
JADE/LEAP [Ber01]	AB	RMI	-	-	LEAP	-	-	MA

Table 3.2: Comparison of middleware approaches

Chapter 4

Coordination Patterns

Traditionally, object oriented languages, use *software design patterns* (see the introduction by Gamma et al. [Gam95]) which provide a reasonable abstraction from the specific problem and allow to model, classify, and compare main characteristics of software structures. In distributed systems, *coordination patterns*, that are, software design patterns for coordination purposes, are used.

In distributed mobile computing scenarios, processes executed on mobile devices are unreliable coordination participants due to, for example, connection losses. In such flexible scenarios, loosely coupled processes are more feasible because they exhibit more potential to continue processing while the coordination partner of interest is not available. Going one step further, in order to study the effect of mobility on distributed computing, coordination scenarios exhibiting different coupling characteristics should be studied. Here, coordination patterns provide a sufficient means to model such distributed scenarios.

Usually, coordination patterns are modeled by applying a template introduced by Gamma et al. [Gam95], pp. 6–8. This template consists of fourteen recommended properties which allow to discuss design patterns in a descriptive manner. Even though this template is very useful and includes a description of patterns by means of source code, neither a precise description on design level, nor a definition of applicable measures is included. This chapter presents a novel extension to the commonly used descriptive modeling framework. Based on the shared *Data Space (DS)* paradigm, each pattern description is extended by a graphical activity model (*Unified Modeling Language (UML) activity diagram*) for concurrent execution, by a sequence of DS primitives, and by measures. These measures allow to describe the coupling between coordinating processes quantitatively and facilitate to classify patterns. The feasibility of the modeling approach is demonstrated by applying the model to eight different coordination patterns which intend to provide a work of reference for the interested reader.

Section 4.1 presents the coordination pattern taxonomy used as well as the

coordination pattern selection criteria. In Section 4.2, the modeling approach is described including the introduction of the measures. Each coordination pattern selected is described in detail in Section 4.3. A comparison of the coordination patterns and a discussion of the modeling approach for mobile distributed systems is presented in Section 4.4.

4.1 Classification and Selection Criteria

Common taxonomies for concurrent processes which interact with one another refer to the aspect of *referential coupling* (or *spatial coupling*) and *temporal coupling*. This taxonomy is used for general distributed systems, like presented by Tanenbaum et al. [Tan02], p. 700, or for mobile agents' coordination, like presented by Cabri et al. [Cab00b]. While *referential coupling* refers to the degree the coordination participants are bound by addresses, *temporal coupling* refers to the degree of synchronization exhibited by the coordinating processes. Figure 4.1 shows four classes derived from this two-dimensional classification scheme. Furthermore, class abbreviations are depicted which are used in the following sections (for example, a coordination pattern referentially coupled but temporally uncoupled is denoted by (r+/t-)).

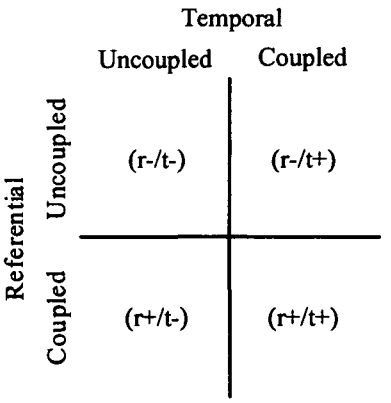


Figure 4.1: Classification of distributed interactions

In shared DS approaches, referential coupling is softened by means of the space abstraction. Messages are not sent to a specific process (identified by an IP-address and a port number), but a logical name is used instead. Furthermore, the shared DS can be used as a persistent storage of messages which exists independently of the coordinating processes. As a consequence, distributed processes may be exchanged easily while the interface to other remote processes does not

change. However, in case one name identifies one particular process this situation is similar to addressing the process according to the message passing paradigm. In this work, the following definition is used for referential coupling:

> A coordination pattern is said to be *referentially coupled* if one or more participating processes perform at least one coordination operation which addresses another particular process explicitly (r+). Otherwise it is said to be *referentially uncoupled* (r-).

Similarly, in this work, the definition for temporal coupling is stated. Since coordination patterns focus on the coordination activities, here, blocking implicitly means that a participant waits for a message from another participant:

> A coordination pattern is said to be *temporally coupled* if one or more participating processes perform at least one blocking coordination operation (t+). Otherwise it is said to be *temporally uncoupled* (t-).

By using coordination patterns, the variety of different coordination activities is reduced to a set of most important scenarios. First, it is necessary to select patterns that are commonly used, well known, and simple enough to be used as basic building blocks for mobile distributed computing scenarios. The following selection criteria are used in order to assure the importance and usefulness of a coordination pattern selection:[1]

- Are the patterns representative (that is, are they most commonly used in literature)?
- Can the patterns be modeled in a unique and precise way?
- Is it possible to classify them according to the taxonomy presented?

Based on these criteria, eight coordination patterns have been selected (Section 4.3 describes the patterns in detail):

Producer/Consumer. The *Producer/Consumer* pattern describes two process types. One process type creates data items and adds them to the shared DS, the other process type consumes data items from the shared DS.

[1] Alternatively, a systematic approach is also possible based on characteristics, like the direction of communication (unidirectional versus bidirectional) or the type and number of coordination primitives used. Because such an approach does not guarantee the importance of the cases under investigation, it has not been used in this work.

Publisher/Subscriber. In the *Publisher/Subscriber* pattern, a subscriber selects events of interest and waits until it is notified about the occurrence of such an event. The publisher process type is responsible for producing events and for generating notifications.

Mailbox. In the *Mailbox* pattern, each process is said to be a peer. Sending a message to a peer means leaving a message to its mailbox. The peer addressed fetches the message in an asynchronous manner.

Master/Worker. In the *Master/Worker* pattern, a master process distributes workload based on a divide-and-conquer strategy, while workers process the tasks and return the results of their work.

Request/Answer. The *Request/Answer* pattern refers to a typical client/server situation, where the client initiates the communication by sending a service request. The client usually blocks until server's answer is received.

Proxy. Using the *Proxy* coordination pattern, two processes coordinate their activities by means of a surrogate process which acts on behalf of one process. The identity of the proxy is known to each process, while the identity of the guarded process is only known to this proxy.

Consensus. The *Consensus* pattern is a rather complex pattern. It describes a scenario where a number of processes have to agree upon a specific subject by proposing values. A distributed consensus algorithms assures the agreement by describing the rounds and types of messages which have to be exchanged reliably.

Broker. The *Broker* pattern describes a client/server system extended by a specific process capable of matching service requests and available servers.

On Coordination Pattern Representativeness

The coordination patterns described are a superset of the coordination patterns and architectures described in single scientific works in the field of distributed computing. The importance of the patterns is either stated implicitly by the selection of the patterns itself, or is stated explicitly. For example, Freisleben et al. [Fre97] emphasize the importance of the *Master/Worker* pattern for parallel computing. Table 4.1 shows and proves that the coordination patterns selected are discussed in various publications on distributed coordination patterns.

However, the work cited includes a few more coordination patterns that have not been selected because they combine several patterns or enhance the patterns only slightly. Haydn et al. [Hay98] discuss additional three other coordination

Pattern Name	Alternative Names and Citations
Producer/Consumer	Producer/Consumer [Küh98b, Küh98a], Generative Communication [Tan02, Gel85]
Publisher/Subscriber	Publisher/Subscriber [Dus03], Meeting Oriented [Cab00b, Tan02], Monitor [Hay98]
Mailbox	Mailbox Coordination [Tan02], Blackboard Based [Cab00b]
Master/Worker	Master/Worker [Fre97], Workflow Manager [Küh98b], Leader Followers - partly [Sch04]
Request/Answer	Request/Answer [Küh98a], Client/Server [Cab00b], Client-Dispatcher-Server [Dus03]
Proxy	Proxy [Dus03, Gam95]
Consensus	Consensus [Cou01], Agreement [Lam82]
Broker	Broker [Hay98, Dus03]

Table 4.1: Coordination patterns in related work

patterns for multi-agent systems named *Embassy-*, *Mediator-*, and *Wrapper* pattern. Since the *Embassy* pattern and the *Wrapper* pattern are adapted and extended *Proxy* patterns, they are not included into the selection. The *Mediator* pattern implements an exchange of conversation rules between two agents by means of a mediator. Hence, this pattern is dedicated to agent specific interactions and is not applicable to coordinating processes in general. Tanenbaum et al. [Tan02], pp. 700–701, and Cabri et al. [Cab00b] use the term *direct coordination* for referentially and temporally coupled patterns. In the latter work, in addition to client/server systems synchronous peer-to-peer systems are named as examples for direct coupling when message passing is assumed. Both works use the term *Linda based systems* when referring to most loosely coupled systems. Since, Linda also supports blocking operations and it is possible to use one template to address a specific process, in this work the term *Linda based* is not used just for this class of pattern. Instead, the entire shared DS approach is Linda based.

Dustar et al. [Dus03], pp. 91–105, extend the pattern selection presented by the *Model View Controller* pattern which has not been included because it focuses more on the separation of different views processes exhibit than on concurrent interactions. Some additional coordination pattern names are defined by Schmidt et al. [Sch04], pp. 365–504: the *Monitor Object*, which is an extended and more complex *Proxy* pattern, the *Half-Sync/Half-Async* pattern, and the

Thread-Specific Storage. The *Half-Sync/Half-Async* pattern describes the combination of synchronous and asynchronous processing parts which is implicitly included in the selection. The *Thread-Specific Storage* concentrates on decoupling of processes by means of separated data spaces. Thus, it is more a means to assure decoupling in various coordination patterns than a separate pattern.

Based on this survey, it is reasonable to state that the coordination patterns selected are representative for concurrent processing and well-known patterns in the field of modern, distributed programming.

On Precise Modeling of Coordination Patterns

The design pattern template introduced by Gamma et al. [Gam95], pp. 6–8, allows to model coordination patterns in a descriptive manner. In addition, UML activity diagrams and sequences of DS primitives are used to model the interactions between the processes in more detail. Section 4.2 furthermore defines new measures which allow to describe the number of DS interactions as well as the coupling between the participants of coordination patterns. Section 4.3 shows the feasibility of the enhanced pattern modeling approach by applying it to the eight coordination patterns selected. On the other hand, Section 4.3 proves that each coordination pattern selected can be described by the model sufficiently.

On Classifying Coordination Patterns

Section 4.3 describes the sequence of shared DS primitives executed by each coordination pattern. Thus, it becomes evident whether the pattern is referentially or temporally coupled by analyzing DS primitive sequences. Section 4.4 summarizes these results and argues the benefits added by the extended pattern modeling approach in terms of classification.

4.2 Coordination Pattern Modeling

Following the software design pattern template which has been introduced by Gamma et al. [Gam95], pp. 6–8, the attributes used to describe the coordination patterns based on the shared DS paradigm are:

Name. This attribute states the name of the coordination pattern.

Classification. Each pattern is assigned to a class based on the taxonomy shown in Figure 4.1.

Intent. The problem addressed by the pattern and the behavior of the pattern is described briefly.

Participants and Structure. The different process types as well as the structural relations between the different process types are described by these attributes.

Motivation, Applicability, and Known Uses. These attributes argue the application of a coordination pattern and describe known scenarios and use cases.

Consequences. Here, pros and cons are discussed as well as trade-offs exhibited by the pattern.

Collaborations. This attribute is used to describe the pattern's control flow extended by a graphical model (UML activity diagram).

Other Names (also known as) and Related Patterns. These attributes discuss other names for the same pattern, variants of the coordination pattern, and relations to other patterns.

Implementation. This attribute is used to describe sequences of DS primitives derived from the UML activity diagrams in detail.

Intentionally, this template differs slightly from the original pattern template. It groups related attributes and leaves out the attribute *sample code* because the description focuses on modeling and not on any specific programming language. UML activity diagrams are chosen for the description of the dynamics and concurrent interactions because they are well-known standardized models widely used to describe co-operative operations of a system, like argued by Hitz et al. [Hit02], pp. 160–161. Besides, they fit more to the shared DS approach than UML interaction diagrams which imply message passing operations.

In addition to the attributes described, measures are applied to the model. Traditionally, measures are used to analyze parallel source code before runtime and parallel programs during runtime. These measures are, for example, the total parallel execution time which considers workload, parallelism, and interaction overhead. Furthermore, the average granularity of programs is analyzed in terms of smallest code segments that should be parallelized. These measures are usually based on specific source code and thus programming languages and runtime environments (for details on the performance of parallel programs see, for example, Hwang et al. [Hwa99], pp. 126–152).

In contrast, here, measures are used to describe the coordination patterns quantitatively based on a particular paradigm (shared DS) and are not based on a single programming language. Consequently, the measures are applicable during software design and, thus, allow the evaluation of the coupling of distributed processes at an early software development phase. The necessity of modeling performance characteristics at a software system design phase is further argued

by the work of Smith et al. [Smi02], pp. 3–25, which introduces the term *Software Performance Engineering*. Here, useful UML extensions are introduced to include performance properties and, for example, to model synchronous and asynchronous message passing [Smi02], pp. 62–67. However, this work does not address shared data spaces.

Co-Primitive	*out()*	*in()*	*inp()*	*rd()*	*rdp()*	*eval()*
Abstract DS Primitive	DS_1	DS_2	DS_3	DS_4	DS_5	DS_6

Table 4.2: Assignment of DS primitives

The measures are based on the DS model introduced in Section 2.3. Table 4.2 shows the assignments between the DS primitives and an abstract set of DS primitive variables DS_i. The following measures are used to describe a coordination pattern:

Number of Concurrent Process Types. P is the set of different concurrent processes P_i, where n is called *dimension* and denotes the set's cardinality and thus the number of concurrent process types:

$$P = \{P_1, ..., P_n\}. \tag{4.1}$$

Number of Concurrent Processes. The number of concurrent processes is determined by a vector $CProc\_Vec$, where each element c_i represents the number of processes in the system for each process type P_i:

$$CProc\_Vec = \begin{pmatrix} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{pmatrix}. \tag{4.2}$$

$CProc$ is the sum of all vector elements and defines the number of concurrent processes in the system:

$$CProc = \sum_{i=1}^{n} c_i. \tag{4.3}$$

Number of Replaceable Concurrent Processes. The term *replaceable process* is used for processes which are not addressed explicitly. For example, in a *Request Answer* scenario, a service request can be processed by multiple (similar) processes while the service result is only dedicated to one client.

Similarly to the number of concurrent processes, the number of replaceable concurrent processes is determined by the vector $CProc\_Rep\_Vec$. Each vector element d_i represents this number of replaceable processes in the system for each process type P_i:

$$CProc\_Rep\_Vec = \begin{pmatrix} d_1 \\ \vdots \\ d_i \\ \vdots \\ d_n \end{pmatrix}. \tag{4.4}$$

$CProc\_Rep$ is the sum of all vector elements and defines the number of replaceable concurrent processes in the coordination pattern. This number is a measure for the degree of uncoupled processes participating. The number can either be used in relation to Equation 4.3 or for comparison between different patterns:

$$CProc\_Rep = \sum_{i=1}^{n} d_i. \tag{4.5}$$

Number of Names/Addresses. The more names (addresses, templates) a coordination pattern uses, the more the participating processes share a priori knowledge. Thus, the number of names used provides an additional measure for referential coupling. This number is denoted by s.

Number of Coordination Operations. The sum of all coordination operations described by a coordination pattern is used to describe the load of interactivity.[2] For each process type P_i the number of coordination operations per space based primitive DS_j is defined by the corresponding element a_{ij} of the matrix COp_Mat. The dimensions of the matrix depend on the used group of DS primitives. For example, using the primitives described in Table 4.2, $m = 6$.

$$COp\_Mat = (a_{ij}), \quad i = 1(1)n, \; j = 1(1)m. \tag{4.6}$$

The number of coordination operations COp describes the frequency of interactions of a pattern and is calculated as the sum of all matrix elements weighted by the number of concurrent processes c_i per process type:

$$COp = \sum_{i=1}^{n} \sum_{j=1}^{m} c_i a_{ij}. \tag{4.7}$$

[2]The sequence of coordination primitives is further translated into corresponding source code.

Number of Blocking Coordination Operations. A subset of the set of coordination operations are the operations which block until the data item of interest is present in the shared DS. This mechanism is used to implement synchronous operations between different processes in shared DS approaches. The number of blocking coordination operations is a measure for the temporal coupling of a pattern which can be related to Equation 4.7 and used for direct comparison with other patterns. In the matrix *COp_Mat* (see 4.6), the columns a_{i2} and a_{i4} describe the number of blocking input operations and blocking read operations ($in()$, $rd()$). Thus, the number of blocking coordination operations *COp_Block* is calculated as follows:

$$COp\_Block = \sum_{i=1}^{n} c_i(a_{i2} + a_{i4}). \tag{4.8}$$

4.3 Description of the Coordination Patterns

In this section, eight selected coordination patterns are described based on the model introduced in Section 4.2. By definition, coordination patterns should be configurable and extensible. Hence, the software patterns described by the UML diagram and the DS primitive sequences are variants. The processes participating in a coordination patterns are termed *process types* and *participants* synonymously. The patterns are described as simple as possible. Thus, no process is further sub-divided into threads. The name of each pattern and its classification is stated in the sub-section's header as defined by Figure 4.1.

In order to keep the measures simple and comparable, some assumptions have been made for all patterns. The variable numbers used to describe the number of times a participant restarts are assumed to be mean values. Furthermore, for all patterns it is assumed that each participant reads or writes only one item to the shared DS per round (for example, a client writes only one request and waits for exactly one answer). In contrast to transformational systems, the coordination patterns described are *reactive systems* using loops to restart their coordination activities again after one round.

4.3.1 Producer/Consumer (r-/t-)

Intent. This general purpose coordination pattern is used for distributed creation, exchange, and consumption of data items. Under the reasonable assumption of a finite shared DS, the processes creating data items compete with one another. The processes execute concurrently, asynchronously, and without a reference identifying the remote processes. In a shared DS,

data items produced and consumed are identified by means of templates or names.

Participants and Structure. Two different process types collaborate, namely the *producer* and the *consumer*. The number of producers is independent from the number of consumers. The message - or data flow between the participants is uni-directional.

Motivation, Applicability, and Known Uses. The *Producer/Consumer* pattern can be used to implement ordered exchange of data and exclusive data consumption and processing. Thus, it can be used for event logging, recording of measurements, for example, in embedded systems, and distributed software installation (see Kühn [Küh98a]).

Consequences. Since the participants do not know each other, they can be exchanged easily. This property addresses maintainability, adaptability, and scalability. Furthermore, the pattern is able to tolerate disconnections, since the participants are not temporally coupled.

Collaborations. The producer connects to the shared DS, generates a data item and adds this item to the DS. The consumer connects to the shared DS, performs a consuming read of a data item identified by a specific name, and processes this item. Figure 4.2 shows the distributed activities by means of an UML activity diagram. All processes restart their activities a number of times. When using decision nodes – which are visualized as diamonds – only the positive branch is included in the diagram and the compensating action is skipped.

Other Names and Related Patterns. This coordination pattern is also known as *generative communication* [Gel85]. Variants of the pattern can be derived by altering some pattern attributes. The semantics of the read operation (consuming or non-consuming), the buffer space size (limited or not limited), the relationship definition between the participants ($1 : 1$, $m : n$, etc.), and the way the participants are coupled may be altered (see also Kühn [Küh98a]). In case consumers do not delete the data items read, these items have to be deleted by other means, like, for example, a garbage collection algorithm. It is most unlikely that temporal coupling is assumed. If the pattern exhibits explicit addressing of a consumer, this pattern becomes referentially coupled.

Implementation (Sequence of DS Primitives). For the two different process types, two different sequences of DS primitives can be derived from the UML activity diagram. Equation 4.9 shows the sequence μ_{P_1} for the producer process type P_1 which executes write operations ($out()$). Equation 4.10 shows the sequence μ_{P_2} for the consumer process type P_2 which

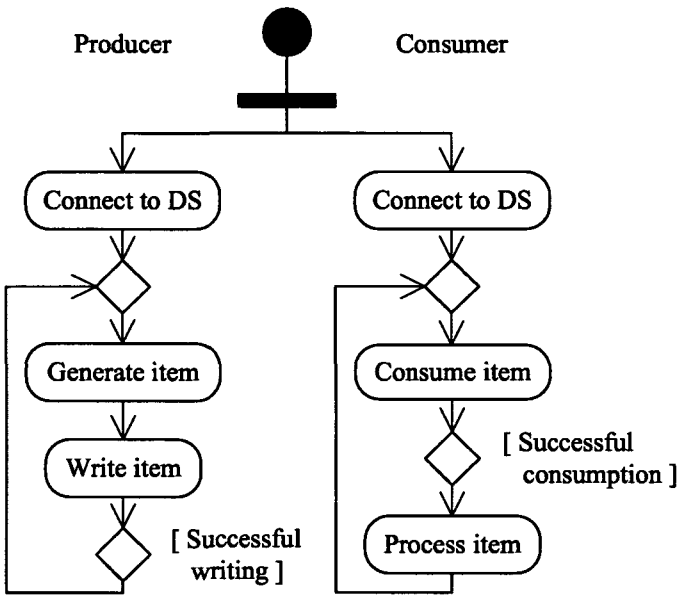


Figure 4.2: Producer/Consumer

executes consuming read operations ($inp()$). Here, k denotes the number of times a producer restarts its task and m denotes the number of times a consumer restarts its task. The processes write to and read from one address (name). In both DS primitive sequences, all primitives are non-blocking although successful reading is only possible if items have already been produced:

$$\mu_{P_1} ::= \overbrace{out(a)}^{k-times}, \text{ where } k \geq 0, \text{ and} \tag{4.9}$$

$$\mu_{P_2} ::= \overbrace{inp(a)}^{m-times}, \text{ where } m \geq 0. \tag{4.10}$$

Measures. The dimension of the *Producer/Consumer (PC)* coordination pattern is:

$$n_{PC} = 2. \tag{4.11}$$

There is no relation between the number of producers and the number of consumers. Additionally, all processes in the system are anonymous and therefore, they are replaceable. The vectors describing the number of processes and the number of replaceable processes per process type are given as follows:

$$CProc\_Vec_{PC} = CProc\_Vec\_Rep_{PC} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}. \tag{4.12}$$

The number of concurrent processes and the number of concurrent replaceable process are calculated as follows:

$$CProc_{PC} = CProc\_Rep_{PC} = c_1 + c_2, \text{ where } c_1, c_2 > 0. \tag{4.13}$$

All processes use only one name to produce and to consume items. Thus, the number of names used evaluates to:

$$s_{PC} = 1. \tag{4.14}$$

The matrix which determines the number of coordination operations of the pattern is given as follows:

$$COp\_Mat_{PC} = \begin{pmatrix} k & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \end{pmatrix}. \tag{4.15}$$

Due to Equation 4.12 and Equation 4.15, the number of coordination operations and the number of blocking coordination operations are calculated as follows:

$$COp_{PC} = c_1 k + c_2 m, \text{ and } COp\_Block_{PC} = 0. \tag{4.16}$$

4.3.2 Publisher/Subscriber (r-/t+)

Intent. A process uses this coordination pattern in order to inform other processes about specific subjects or changes (in general, events). The receivers need to be informed in (soft) real-time. Instead of requesting the status from the remote process actively by means of polling, the remote process notifies the interested group of processes. This pattern can be used to synchronize states among distributed processes.

Participants and Structure. This coordination pattern consists of two process types, one is responsible for publishing events (*publisher*) while the other process is ready to receive events which it has previously subscribed to (*subscriber*). The subscribers do not coordinate their actions. The number of subscribers and the number of publishers are independent of one another. In a shared DS, the space is used by subscribers to place a subscription and by publishers to generate notifications (that are, data items) for each subscriber which has previously subscribed to this event. One process can subscribe at multiple publishers and one process can act both as a subscriber and as a publisher. However, the pattern variant described here does not allow these two special cases. The data flow exhibited by this pattern is bi-directional.

Motivation, Applicability, and Known Uses. The *Publisher/Subscriber* pattern provides an efficient technique to notify a group of processes that may change over time. For example, notification of remote GUIs is one of the application areas, in particular whenever multiple GUIs are involved which are not known to the publisher a-priori. Additionally, notification about sensors' state changes in distributed embedded systems can be realized by means of this pattern.

Consequences. The subscriber processes have to wait for a notification of the publishers, and thus, they can only provide a reliable service when the notification succeeds. In case no acknowledgments are sent to assure that each subscriber has received a notification, at most once semantics can be achieved.

Collaborations. Figure 4.3 shows the distributed activities of the two process types by means of an UML activity diagram. Here, one publisher is responsible for publishing events of one event type. The subscriber generates subscriptions and then waits for events to consume. After consuming these events, the subscriber processes them and restarts by waiting for new events. The publisher process generates an event, reads subscriptions in a non-consuming way, and notifies subscribers by writing of events. Both participants restart their activities a number of times. Note, that this pattern does not contain any means for deleting a subscription from the list.

Other Names and Related Patterns. This coordination pattern is also known as *Observer* and *Dependents*. Two variants can be derived by altering the information retrieval strategies: the *push model* or the *pull model*. While in the first model, the publisher sends all relevant information within one single message (data item or notification), the second model relaxes this concept. Here, the publisher just notifies the subscribers that some change has happened. The content of the status change can be requested on demand. In the case presented, the publisher notifies the subscribers by writing a single data item that notifies the subscribers to read this item. Thus, from a modeling point of view, a push model is implemented because the events' content included in the notification. The coordination pattern can be further extended, for example, by a registration process for publishers, by acknowledges sent from subscribers to publishers, and by garbage collection mechanisms as proposed by [Gro00] for CORSO [Küh01]. Other variants are the *Gatekeeper*, a process that filters incoming events and produces outgoing events for a group of subscribed processes. The pattern variant based on the DS paradigm is similar to a variant based on *event channels*. Such channels allow to generate events and to receive events

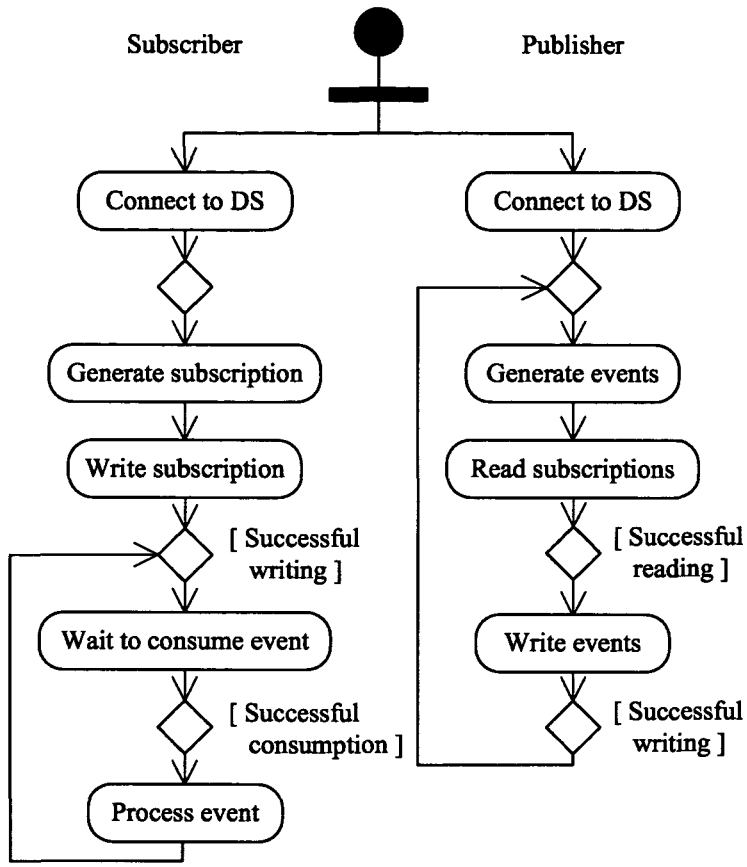


Figure 4.3: Publisher/Subscriber

anonymously. For a detailed discussion of related patterns, see Dustar et al. [Dus03], pp. 103–105.

Implementation (Sequence of DS Primitives). Since this coordination pattern consists of two different process types, two different sequences of DS primitives are derived from the UML activity diagram. The publisher performs a non-consuming read of subscriptions (*rdp*()) and publishes its events to shared DS using a specific name (*out*()). Equation 4.17 shows the sequence μ_{P_1} for the publisher process types P_1. The subscriber performs a write operation (*out*()) in order to place its subscription followed by synchronous and consuming read operations for the events published (*in*()). Equation 4.18 shows the sequence μ_{P_2} for the subscriber process type P_2. Here, m denotes the number of times the publisher restarts its

task and k denotes the number of times the subscriber restarts its task:

$$\mu_{P_1} ::= \overbrace{\overbrace{rdp(a)}^{p-times}.\overbrace{out(b)}^{p-times}}^{m-times}, \text{ where } m,p \geq 0, \text{ and} \tag{4.17}$$

$$\mu_{P_2} ::= out(a).\overbrace{in(b)}^{k-times}, \text{ where } k \geq 0. \tag{4.18}$$

Measures. The dimension of the *Publisher/Subscriber (PS)* pattern is:

$$n_{PS} = 2. \tag{4.19}$$

The number of subscribers is independent of the number of publishers. All processes are replaceable. For the subscribers, this simply implies that they fail silently and another subscriber may take over. The following vectors describe the number of concurrent processes and the number of replaceable concurrent processes per process type:

$$CProc\_Vec_{PS} = CProc\_Vec\_Rep_{PS} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}. \tag{4.20}$$

The number of concurrent processes and the number of replaceable concurrent processes are calculated as follows:

$$CProc_{PS} = CProc\_Rep_{PS} = c_1 + c_2, \text{ where } c_1, c_2 > 0. \tag{4.21}$$

The *Publisher/Subscriber* coordination pattern uses one known name for exchanging subscriptions and one name for writing events to. Thus,

$$s_{PS} = 2. \tag{4.22}$$

The matrix which determines the number of coordination operations is given as follows:

$$COp\_Mat_{PS} = \begin{pmatrix} mp & 0 & 0 & 0 & mp & 0 \\ 1 & k & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.23}$$

Since a subscriber has to wait for the events of providers available, m and k are dependent. Furthermore, p evaluates to the number of subscriptions available, that is, number of subscribers:

$$k = c_1 m, \text{ and } p = c_2. \tag{4.24}$$

Due to Equation 4.20, Equation 4.23, and Equation 4.24, the number of coordination operations and the number of blocking coordination operations are calculated as follows:

$$COp_{PS} = c_2(1 + 3k), \text{ and } COp\_Sync_{PS} = c_2 k. \tag{4.25}$$

4.3.3 Mailbox (r+/t-)

Intent. This coordination pattern is used by processes in order to store and retrieve messages via the shared DS. Each process can be addressed explicitly by a specific name or mailbox in the DS which stores the messages persistently. It is not important to know, where the processes currently execute or when they read the messages (asynchronous communication).

Participants and Structures. The *Mailbox* pattern consists of only one process type, a *peer*. The number of peers is arbitrary and the data flow is bi-directional.

Motivation, Applicability, and Known Uses. This coordination pattern is applicable to all kind of direct but asynchronous communication between processes where no central control is needed and no pre-defined role concept is applicable. Asynchronous *peer-to-peer* applications as well as persistent messaging systems are examples for this pattern. When the coordination partners are not known to each other a priori, then advertisement and lookup have to be implemented to enable coordination activities.

Consequences. A trade-off is identified between the benefits of anonymity in terms of reconfigurability and the overload caused by lookup and registration algorithms.

Collaboration. Each peer connects to the DS, generates a new message and writes it to the mailbox of a specific remote peer if necessary. Then, the peer consumes and processes an item, that is, a received message, which is retrieved from its own mailbox. Figure 4.4 shows the activity flow of a peer which restarts its task a number of times. Here, although this is a restriction of freedom, the peers are assumed to write a message and to try to read a message during each round in order to model a highly interactive pattern.

Other Names and Related Patterns. This pattern is also known as *Blackboard Based Coordination* in the agent research community, as described by Cabri et al [Cab00b]. Here, messages are written to a blackboard (that is, a shared DS) independent of the current place or state of the receiver agent. Tanenbaum et al. [Tan02], pp. 700–701, refer to this pattern as *Mailbox Coordination* which is closely related to persistent message-oriented communication.

Implementation (Sequence of DS Primitives). This coordination pattern consists of one process type which performs *out*() operations in order to store a message for a remote peer addressed persistently via the shared

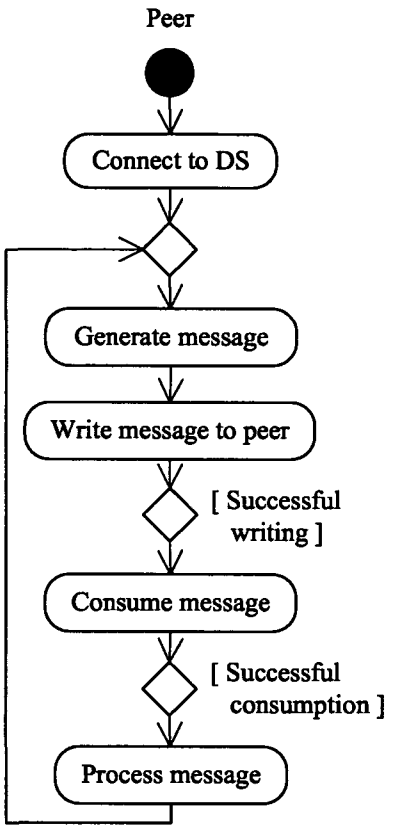


Figure 4.4: Mailbox

DS. Then, the peer consumes one message from its own mailbox in a non-blocking manner ($inp()$). Equation 4.26 shows the sequence μ_{P_1} for a peer P_1. Here, k denotes the number of times a peer restarts its task (note, that the name the process type writes to a_{P_1} has to be a valid mailbox):

$$\mu_{P_1} ::= \overbrace{out(a_{P_1}).inp(b_{P_1})}^{k-times}, \text{ where } k > 0. \tag{4.26}$$

Measures. The dimension of the *Mailbox (Mb)* coordination pattern is given as follows:

$$n_{Mb} = 1. \tag{4.27}$$

Since the pattern exhibits only one dimension, the vector describing the number of concurrent processes consists of only a single entry. No peer is replaceable in this pattern:

$$CProc\_Vec_{Mb} = (\ c_1\ ), \text{ and } CProc\_Vec\_Rep_{Mb} = (\ 0\ ). \tag{4.28}$$

Thus, the number of concurrent processes and the number of replaceable concurrent processes are calculated as follows:

$$CProc_{Mb} = c_1, \text{ and } CProc\_Rep_{Mb} = 0. \tag{4.29}$$

The pattern uses one known name per peer for storing messages. Thus,

$$s_{Mb} = c_1. \tag{4.30}$$

The matrix which determines the number of coordination operations reduces to the following vector:

$$COp\_Mat_{Mb} = \begin{pmatrix} k & 0 & k & 0 & 0 & 0 \end{pmatrix}. \tag{4.31}$$

Hence, the number of coordination operations and the number of blocking coordination operations are calculated as follows (based on Equation 4.28 and Equation 4.31):

$$COp_{Mb} = 2c_1k, \text{ and } COp\_Block_{Mb} = 0. \tag{4.32}$$

4.3.4 Master/Worker (r+/t+)

Intent. The intent of this coordination pattern is to divide a task into sub-tasks and force other processes to execute these sub-tasks concurrently. A centralized coordination task is responsible for collecting the results of the sub-tasks, which usually means a last computational step in order to achieve the final result.

Participants and Structure. Two types of different processes are described by the pattern. The *master* process is a component which distributes sub-tasks, collects single results, and merges these results. Hence, the master can also be seen as a workload generator. Multiple masters here are used to model the situation of multiple workload sources. The other process type is termed *worker*. This process executes a sub-task and produces a result. The workers usually work on independent tasks, that is, on tasks that need no further coordination effort. One name is used by the master for writing orders to the DS and another name per master is used by the worker process type to write the results to. The data flow is bi-directional.

Motivation, Applicability, and Known Uses. This coordination pattern is most useful in situations where it is possible to distribute workload. In the field of parallel computing, the pattern is used in order to increase performance of computationally extensive tasks. In fault-tolerant computing scenarios, this pattern can be used to replicate computation.

Consequences. On one hand, the coordination pattern exhibits the possibility to assure fault tolerance based on redundant computation and speedup of high-performance computing task. However, the decomposition of the task may lead to significant coordination overhead, which may decrease the possible speed up.

Collaborations. Figure 4.5 shows the distributed activities by means of an UML activity diagram. Both process types loop a number of times. After writing a new order to the space, the master tries to collect a new result currently available in the shared DS in a non-blocking manner. This sequence has been chosen to simplify possible rounds where a process generates only a new order or attempts only to collect a result. Every worker starts with retrieving a new order from the shared DS, processes the order, and submits the result.

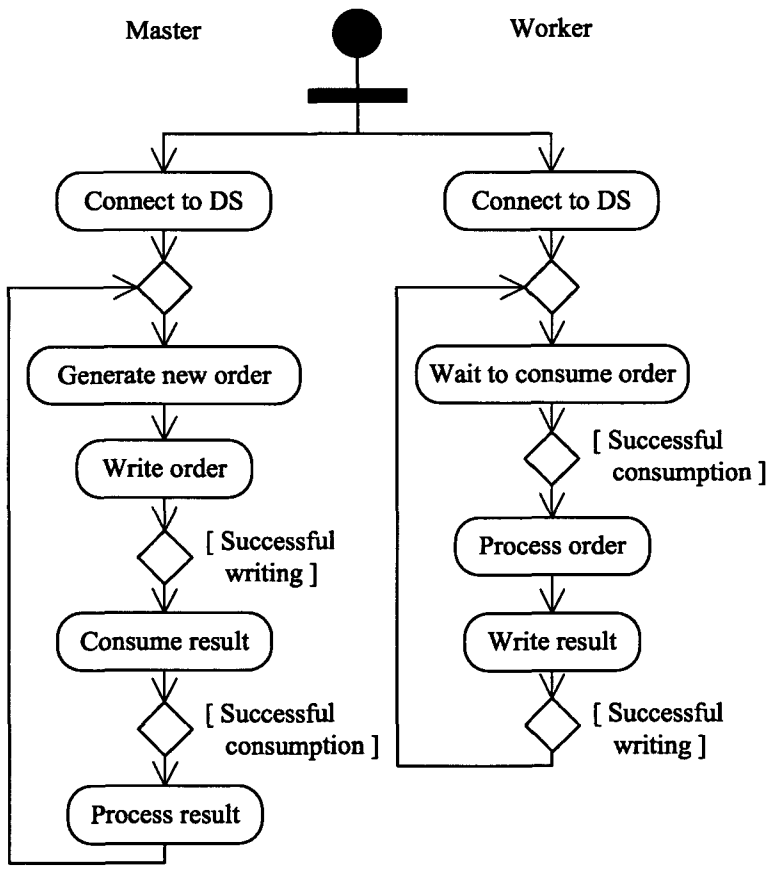


Figure 4.5: Master/Worker

Other Names and Related Patterns. This well-known coordination pattern

is also known as *Master/Slave*, *Manager/Worker*, or *Workflow Manager*. A variant can be derived when assuming synchronous operation, that is, the master waits for the results of the workers. Other variants depend on the task the master has to execute (see Dustar et al. [Dus03], pp. 97–100). For example, in case of using this pattern to replicate a single computation for fault tolerance purposes, the result of one worker may be sufficient for the success of the task (or, in general, k out of n results). In contrast, assuming a typical divide-and-conquer scenario where the original task is divided into sub-tasks processed by workers, every sub-task has to finish correctly.

Implementation (Sequence of DS Primitives). The master P_1 performs a write operation (*out*()) followed by a consuming non-blocking read operation (*inp*()). The worker P_2 performs a blocking consuming read operation (*in*()) followed by a write operation(*out*()). Equation 4.33 and Equation 4.34 show the primitives sequences for the master (μ_{P_1}) and the worker (μ_{P_2}) respectively. Here, k denotes the number of times a master restarts its task and m denotes the number of times a worker restarts its task:

$$\mu_{P_1} ::= \overbrace{out(a).inp(b_{P_1})}^{k-times}, \text{ where } k \geq 0, \text{ and} \tag{4.33}$$

$$\mu_{P_2} ::= \overbrace{in(a).out(b_{P_1})}^{m-times}, \text{ where } m \geq 0. \tag{4.34}$$

Measures. The dimension of the *Master/Worker (MW)* pattern is:

$$n_{MW} = 2. \tag{4.35}$$

The number of master processes and the number of worker processes are independent of one another and all worker processes are replaceable. Thus, the vectors describing the number of concurrent processes in the system per process type are given as follows:

$$CProc\_Vec_{MW} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, \text{ and } CProc\_Vec\_Rep_{MW} = \begin{pmatrix} 0 \\ c_2 \end{pmatrix}. \tag{4.36}$$

The number of concurrent processes and the number of replaceable concurrent processes are calculated as follows:

$$CProc_{MW} = c_1 + c_2, \text{ and } CProc\_Rep_{MW} = c_2. \tag{4.37}$$

For the bi-directional data flow, one name is used to write orders to the DS and one name per master process is used for the results. Thus,

$$s_{MW} = 1 + c_1. \tag{4.38}$$

The matrix which determines the number of coordination primitives is given as follows:

$$COp\_Mat_{MW} = \begin{pmatrix} k & 0 & k & 0 & 0 & 0 \\ m & m & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.39}$$

Since the workers perform blocking consuming read operations, the number of loops carried out depends on the number of times the master processes restart. Accordingly, it follows:

$$c_1 k = c_2 m. \tag{4.40}$$

Due to Equations 4.36, 4.39, and 4.40), the number of coordination operations and the number of blocking coordination operations are calculated as follows:

$$COp_{MW} = 4c_1 k, \text{ and } COp\_Block_{MW} = c_1 k. \tag{4.41}$$

4.3.5 Request/Answer (r+/t+)

Intent. This coordination pattern occurs in distributed systems whenever a process requires a service provided by another remote process. The process generates a service request and waits for the answer.

Participants and Structure. The *Request/Answer* pattern consists of two different process types, a *client* (or requester) and a *server*. The number of the servers available and clients participating is not known by the processes. The coordination pattern implements a bi-directional data flow.

Motivation, Application and Known Uses. This well-known coordination pattern is used whenever services are needed by a group of processes which may change dynamically. It is implemented in traditional client/server systems. If the name of the server is not known to the clients, dispatchers or name services may manage access to the server although its identity is hidden. In shared DS, the space itself implements the function of such a name server. Typical examples are flight reservation and hotel booking scenarios (see the examples presented by Kühn [Küh98a]).

Consequences. Because clients wait for results, they are vulnerable to faults caused by the remote service providers. In case of faults, they cannot proceed with their tasks.

Collaborations. Figure 4.6 shows the distributed activities graphically by means of an UML activity diagram. The client initiates the coordination activities by writing a request to the shared DS identified by a name. The server task waits to consume a request and processes the service requested. The server's result is written to the DS using a name assigned to the client.

Both processes loop a number of times. Note, that the request is usually processed by a new thread of the server.

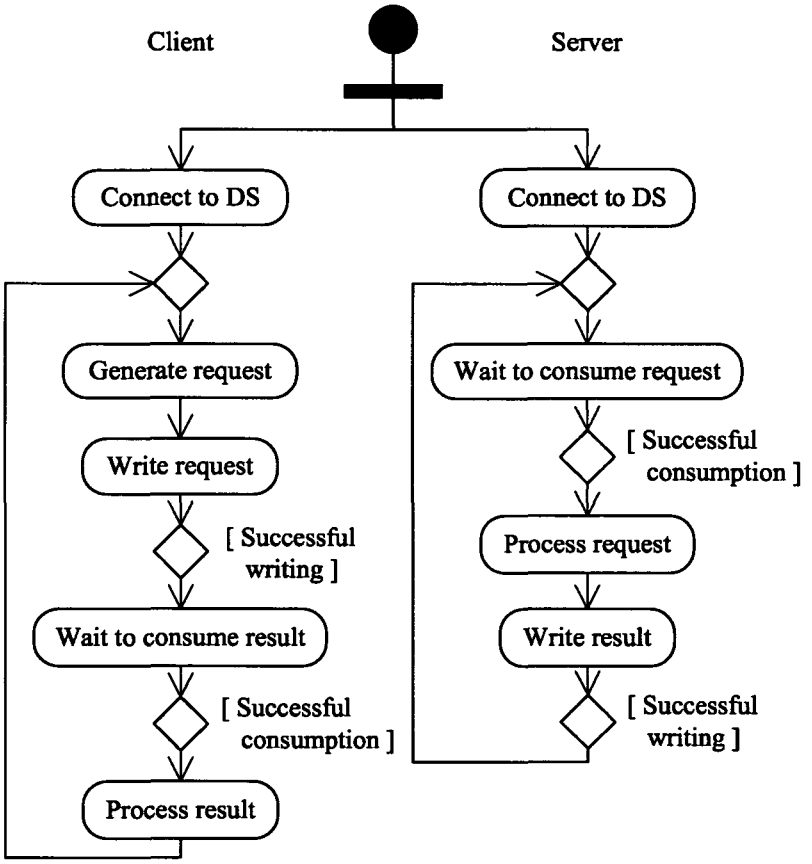


Figure 4.6: Request/Answer

Other Names and Related Patterns. *Client/Server* pattern is another name of this coordination pattern. Dustar et al. [Dus03] propose a variant termed *Client-Dispatcher-Server*, where a dispatcher is used as a name service. Other variants can be derived in terms of number of servers and the way they distribute workload. Another variant uses the spawning primitive of the DS in order to launch a remote server as described by Kühn [Küh98a]. This variant implements the *Remote Procedure Call (RPC)* for shared DS.

Implementation (Sequence of DS Primitives). From the UML activity diagram, the sequences of DS coordination primitives are derived. The client sends ever new requests and waits for the answers. Hence, the client first performs a write operations (*out*()) followed by a blocking consuming read

operation ($in()$). Equation 4.42 shows the sequence μ_{P_1} for the client process type P_1. The server performs a blocking consuming read of a new request ($in()$) and writes the answer to a name in the shared DS assigned to the client ($out()$). Equation 4.43 describes the sequence μ_{P_2} for the server process type P_2. Here, k denotes the number of times a client restarts its task and m denotes the number of times a server restarts its task:

$$\mu_{P_1} ::= \overbrace{out(a).in(b_{P_1})}^{k-times}, \text{ where } k \geq 0, \text{ and} \tag{4.42}$$

$$\mu_{P_2} ::= \overbrace{in(a).out(b_{P_1})}^{m-times}, \text{ where } m \geq 0. \tag{4.43}$$

Measures. The dimension of the *Request/Answer (RA)* pattern is:

$$n_{RA} = 2. \tag{4.44}$$

The number of clients is independent of the number of servers. Because each client waits for an answer written to a name identifying the client, clients are not replaceable. The following vectors describe the number of concurrent clients and concurrent servers and their replaceable counterpart:

$$CProc\_Vec_{RA} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, \text{ and } CProc\_Vec\_Rep_{RA} = \begin{pmatrix} 0 \\ c_2 \end{pmatrix}. \tag{4.45}$$

Thus, the number of concurrent processes and the number of replaceable concurrent processes are calculated as follows:

$$CProc_{RA} = c_1 + c_2, \text{ and } CProc\_Rep_{RA} = c_2. \tag{4.46}$$

The pattern uses one known name for requests and one name for each client. Thus,

$$s_{RA} = 1 + c_1. \tag{4.47}$$

The matrix which determines the number of coordination operations is given as follows:

$$COp\_Mat_{RA} = \begin{pmatrix} k & k & 0 & 0 & 0 & 0 \\ m & m & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.48}$$

Since a client has to wait for an answer from the servers available, m and k are dependent:

$$c_1 k = c_2 m. \tag{4.49}$$

Hence, the number of coordination operations and the number of blocking coordination operations are calculated as follows (based on Equations 4.45, 4.48, and 4.49):

$$COp_{RA} = 4kc_1, \text{ and } COp\_Block_{RA} = 2kc_1. \tag{4.50}$$

4.3.6 Proxy (r+/t+)

Intent. The *Proxy* pattern intends to provide a representative or surrogate for a process. Other remote processes may communicate with the proxy instead of the guarded process without changing their interfaces, that is, the communication and coordination protocol. The proxy process may offer additional application specific services, like, for example, access control. Since the *Proxy* pattern does not define the purpose and sequence of collaboration between the participants, here, the simple *Mailbox* coordination pattern is chosen.

Participants and Structure. The *Proxy* pattern consists of three process types: a *proxy* process, a peer guarded by the proxy (*proxy-peer*) and another peer which is not guarded by a proxy (*peer*). The usual case of a one-to-one relationship (1 : 1) between a proxy-peer and its proxy is assumed. The number of peer processes is independent of the number of proxy/proxy-peer pairs. The data flow between the process types is bidirectional – although not necessarily between the same processes.

Motivation, Applicability, and Known Uses. One main reason for using a proxy process lies in separating the interface to a peer process from the core functions of the process. Additionally, proxies may add functions like access control, synchronization mechanisms, and the concealment of the remoteness of a process. For example, the pattern is applicable in case a local representative for a remote object is required (*remote proxy*, described by Gamma et al. [Gam95], pp. 91–105). A *protection proxy* can be used to control access to the original process (or object) by adding additional filtering functions. In firewall approaches, proxies are used to hide the existence of a firewall from the distributed system.

Consequences. This coordination pattern exhibits benefits in terms of security and decoupling of processes. Drawbacks of this pattern are the overhead added. In particular in shared DS, some functions, like access control, may be implemented by the DS instead.

Collaborations. A peer may address a proxy-peer not directly. Instead, the proxy is addressed which waits for consuming a new message. The proxy processes the message and writes it to the mailbox of the proxy-peer addressed. The proxy-peer acts like any other peer. It consumes the message from its mailbox and writes a new messages to an arbitrary other peer. Figure 4.7 shows the distributed activities by means of an UML activity diagram.

Other Names and Related Patterns. This coordination pattern is also known as *Surrogate* (see Gamma et al. [Gam95], pp. 207–217). A vari-

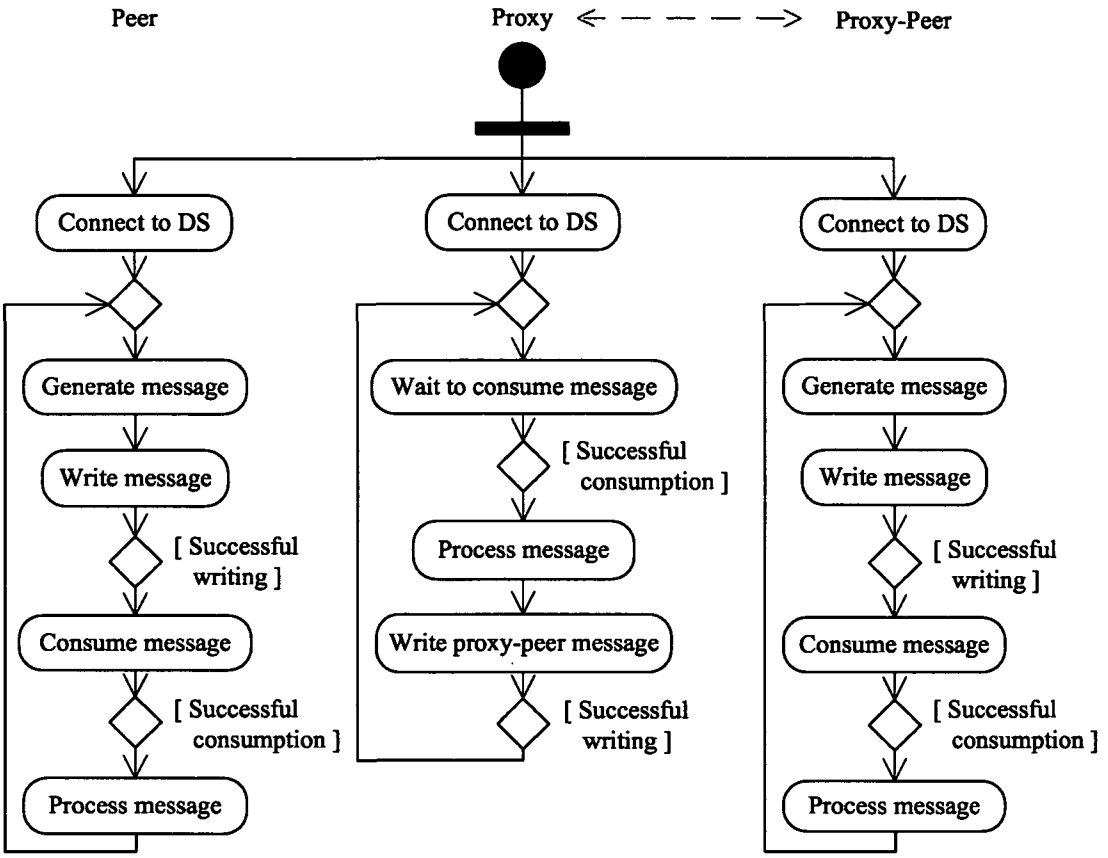


Figure 4.7: Proxy

ant of the *Proxy* pattern can be derived when one proxy serves more than one proxy-peer. Furthermore, the intent of the proxy pattern may vary based on the application. Gamma et al. describe two additional variants: the *Adapter* pattern and the *Decorator* pattern. In contrast to the proxy, an adapter alters the interface of the proxy process (or object). In addition to providing access to the (hidden) peer, a decorator adds responsibilities to the original process (or object). From a computer language perspective, the decorator provides similar mechanisms than inheritance. Here, if not prohibited by protection flags, a sub class includes all functions of the super class and may consist of additional functions.

Implementation (Sequence of DS Primitives). Three different participants are described by the pattern (although the proxy-peer and the peer execute the same operations). The peer and the proxy-peer perform *out*() operations in order to write messages for another peer persistently to the shared DS. Then, the peer and the proxy-peer consume the messages in

their own mailbox using a non-blocking consuming read operation (*inp*()). The proxy process reads a message in a blocking and consuming manner (*in*()) and forwards it to the guarded proxy-peer (*out*()). Equation 4.51 shows the sequence μ_{P1} for the peer P_1, Equation 4.53 shows the sequences for the proxy-peer P_3 and Equation 4.52 shows the sequence μ_{P_2} for the proxy P_2. The number of times the peer and the proxy-peer restart their tasks is denoted by k.[3] For the proxy, m denotes the number of times this process type restart its task:

$$\mu_{P_1} ::= \overbrace{out(a_{P_2}).inp(b_{P_1})}^{k-times}, \text{ where } k \geq 0, \tag{4.51}$$

$$\mu_{P_2} ::= \overbrace{in(a_{P_2}).out(c_{P_3})}^{m-times}, \text{ where } m \geq 0, \text{ and} \tag{4.52}$$

$$\mu_{P_3} ::= \overbrace{out(b_{P_1}).inp(c_{P_3})}^{k-times}. \tag{4.53}$$

Measures. The dimension of the *Proxy (Pr)* pattern is:

$$n_{Pr} = 3. \tag{4.54}$$

While the number of peers is independent of the number of other process types, the number of proxy-peers is the same as the number of proxies ($c_2 = c_3$). Since no processes are replaceable, the vectors describing the numbers of concurrent processes per process type are given as follows:

$$CProc\_Vec_{Pr} = \begin{pmatrix} c_1 \\ c_2 \\ c_2 \end{pmatrix}, \text{ and } CProc\_Vec\_Rep_{Pr} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.55}$$

Thus, the number of concurrent processes and the number of replaceable concurrent processes is calculated as follows:

$$CProc_{Pr} = c_1 + 2c_2, \text{ and } CProc\_Rep_{Pr} = 0. \tag{4.56}$$

The coordination pattern uses one name per peer (including the proxy-peer) and one name per proxy. Thus,

$$s_{Pr} = c_1 + 2c_2. \tag{4.57}$$

The matrix which determines the number of coordination operations is given as follows:

$$COp\_Mat_{Pr} = \begin{pmatrix} k & 0 & k & 0 & 0 & 0 \\ m & m & 0 & 0 & 0 & 0 \\ k & 0 & k & 0 & 0 & 0 \end{pmatrix}. \tag{4.58}$$

[3]The proxy-peer process type performs the same actions than the peer process type. Hence, the number of loops is the same.

Since the proxy performs a consuming read and thus, it waits for data items from a peer, m and k are dependent:

$$c_1 k = c_2 m. \tag{4.59}$$

Due to Equations 4.55, 4.58, and 4.59, the number of coordination operations and the number of blocking coordination operations are calculated as follows:

$$COp_{Pr} = 2k(2c_1 + c_2), \text{ and } COp\_Block_{Pr} = kc_1. \tag{4.60}$$

4.3.7 Consensus (r+/t+)

Intent. The *Consensus* pattern describes a situation in which all participants have to agree on a certain subject by proposing values and by executing a specific agreement protocol. In asynchronous systems without further assumptions, it is not possible to implement distributed consensus because processes may take arbitrary time intervals to answer (for a detailed discussion, see Fischer et al. [Fis85]). The requirements of a consensus algorithm are stated by Coulouris et al. [Cou01], p. 452, as: (i) *termination*, (ii) *agreement*, and (iii) *integrity*. Here, termination means, that each participating process makes its decision eventually. The agreement property states, that the decision of all correct processes should be the same. Finally, integrity means, that in case all processes propose the same value, then all processes should agree on that value.

Participants and Structure. This pattern consists of one process type denoted as *participant*. In this work, a decentralized agreement algorithm is proposed which is based on a priori knowledge about the consensus group members. Termination can be assured by means of timers and error states in case not all expected values are received in time. Agreement and integrity are achieved because of the algorithm proposed: Either all processes calculate the consensus value similarly (that is, based on the same algorithm) or none of them. Other failure compensation or voting strategies are not supported by this consensus variant. The data flow is bi-directional for this pattern.

Motivation, Applicability, and Known Uses. This coordination pattern is used whenever a group of processes have to agree on a shared value. For example, this pattern is applicable in election algorithms (see Coulouris et al. [Cou01], pp. 431–436), for voting between replicated components in order to locate malfunctioning components, and, in embedded systems, for decision support on different sensor measure values. For example, Kopetz [Kop97], pp. 111–117, discusses replica determinism.

Consequences. In scenarios with faulty components additional protocol overhead is necessary to reach an agreement which slows down the algorithm. Additionally, in the worst case, an agreement may not be achievable. Lamport et al. [Lam82] discuss these problems in their work on the *Byzantine Generals Problem.*

Collaborations. The algorithm used assures the agreement after one round in a fault-free scenario. For simplicity reason, the number of participants and the names of the communication variables have to be known a priori. Therefore, the algorithm uses two different flags and the voting value to assure that all participants have passed a critical section (phase):

Ready Flag. The ready flag is written by each process in case this process is ready to start a new consensus round.

Value. The value represents the vote of the process which is compared to the other votes by a single algorithm executed by each process. Writing the value signals that the process is ready to read the votes of all other processes.

Finished Flag. The finished flag is used to signal that all votes have been read and, thus, each process is ready to calculate the decision value (that is, the consensus has been reached).

Figure 4.8 shows the UML activity diagram of the coordination pattern. Each participant notifies the other processes that it is ready to start (*Write own ready flag*) and waits for the same information retrieved from the other processes (*Wait to read all ready flags*). Then the participant consumes a former generated flag which is used for signaling the end of a consensus round (*Consume own finished flag*), if such a flag exists. It generates a new value and writes this value to a specific address of the shared DS (*Write own value*). During the next step, the participant waits for the values of the other participants (*Wait to read all values*). At this step, the process has to consume its ready flag in order to prevent the restart of the agreement before the process is ready again (*Consume own ready flag*) caused by the old ready flag. It signals the other processes that it has passed reading (*Write own finished flag*) and waits for all others to finish (*Wait to read all finished flags*). During the last steps, the process consumes its own value (*Consume own value*) and calculates the decision value. Here, each participant restarts its task a number of times.

Other Names and Related Patterns. This pattern is also termed *consensus problem* or *problem of agreement* by Coulouris et al. [Cou01], p. 451–453. Variants of the pattern can be derived by applying it to different consensus problems. For example, the agreement on a vector consisting of one

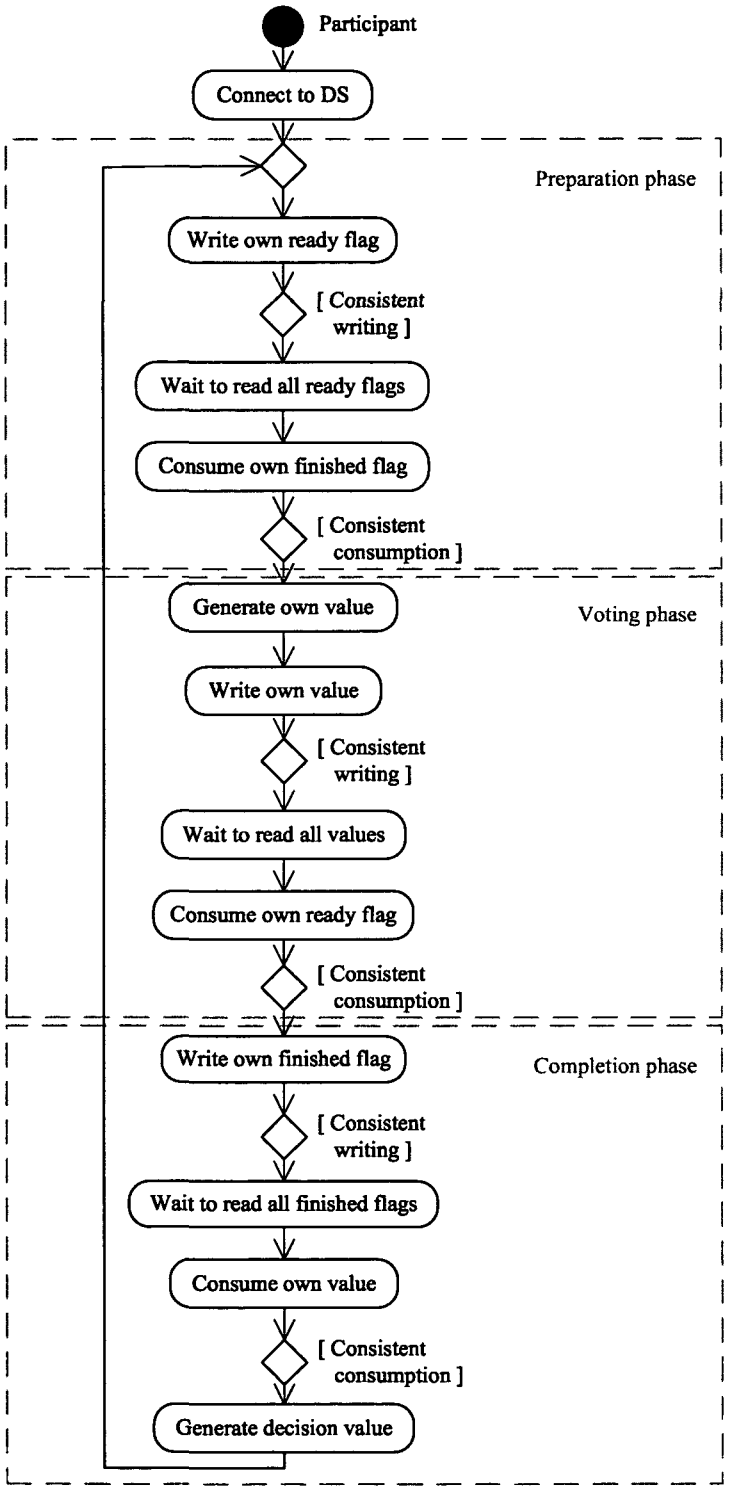


Figure 4.8: Consensus

result per participant is termed *interactive consistency*. Furthermore, different variants can be derived by using a centralized version, by adding fault tolerance (that is, tolerating a number of participants to fail), and by relaxation of referential coupling.

Implementation (Sequence of DS Primitives). This decentralized coordination pattern consists of only one process type. However, a coordination sequence makes only sense in case there are at least 2 (similar) processes participating. The UML activity diagram of Figure 4.8 is transformed into a sequence of DS primitives by using an *out*() operation for each writing, the *inp*() operation for a consuming non-blocking read, and the *rd*() operation for a blocking non-consuming read of items. Equation 4.61 shows the sequences μ_{P_1} for each participant P_1. Here, k denotes the number of times the participant restarts its task and p denotes the number of values to expect:[4]

$$\mu_{P_1} ::= \overbrace{preparation.voting.completion}^{k-times}, \text{ where } k \geq 0, \tag{4.61}$$

$$preparation ::= out(r).\overbrace{rd(r_{P_1})}^{p-times}.inp(f_{P_1}),$$

$$voting ::= out(v_{P_1}).\overbrace{rd(v_{P_1})}^{p-times}.inp(r_{P_1}), \text{ and}$$

$$completion ::= out(f_{P_1}).\overbrace{rd(f_{P_1})}^{p-times}.inp(v_{P_1}), \text{ where } p > 0.$$

Measures. The dimension of the *Consensus (Cs)* pattern is given as follows:

$$n_{Cs} = 1. \tag{4.62}$$

The vector describing the number of concurrent processes per process type exhibits only one dimension and no process type is replaceable. Thus,

$$CProc\_Vec_{Cs} = (\ c_1\ ), \text{ and } CProc\_Vec\_Rep_{Cs} = (\ 0\ ). \tag{4.63}$$

The number of concurrent processes and the number of replaceable concurrent processes are calculated as follows:

$$CProc_{Cs} = c_1, \text{ and } CProc\_Rep_{Cs} = 0. \tag{4.64}$$

[4]Without loss of generality, it is assumed that at least one other process is participating.

The coordination pattern uses three different names per participant. Thus,

$$s_{Cs} = 3c_1. \tag{4.65}$$

The matrix which determines the number of coordination operations is given as follows:

$$COp\_Mat_{Cs} = \begin{pmatrix} 3k & 0 & 3k & 3kp & 0 & 0 \end{pmatrix}. \tag{4.66}$$

Since the number of values generated during one round is equal to the number of other participants, the following equation is derived:

$$p = c_1 - 1, \quad \text{where } c_1 > 1. \tag{4.67}$$

Hence, the number of coordination operations and the number of blocking coordination operations are calculated as follows (based on Equations 4.63, 4.66, and 4.67):

$$COp_{Cs} = 3c_1k(1 + c_1), \quad \text{and} \quad COp\_Sync_{Cs} = 3c_1k(c_1 - 1). \tag{4.68}$$

4.3.8 Broker (r+/t+)

Intent. The *Broker* pattern provides the decoupling of client processes and servers (service providers). The broker process receives requests and tries to match these requests with known and available servers. Usually, servers register with the broker process before they are ready for service requests.

Participants and Structure This coordination pattern consists of three process types: a *client*, a *server*, and a *broker*. The client and the server behave like described by the *Request/Answer* pattern, except that a broker is needed to inform the client about the reference it should use for requests. The data flow is bi-directional for all processes.

Motivation, Application, and Known Uses. The pattern is most useful in scenarios where clients do not exactly know the address of a server available, because, for example, addresses change frequently. Here, the central role of the broker helps to find the right server efficiently. In particular, systems with a multitude of servers and clients benefit from this pattern because it allows to reduce message overload caused by decentralized service discovery mechanisms. A well known example of a broker is the *Object Request Broker (ORB)* included in the *Common Object Request Broker Architecture (CORBA)* of the *Object Management Group (OMG)* [OMG04a]. In multi-agent systems, a broker may hold a complex capability model for each service provider and apply extensive search algorithms (see the description in Hayden et al. [Hay98]).

Consequences. The main advantages of this pattern are *call transparency* and, thus, *reusability*, and the avoidance of message flooding for lookup purposes. Possible disadvantages are, performance decrease because of the indirection caused by the broker process types and a single point of failure in the simple centralized case.

Collaborations. The *Broker* pattern is similar to the *Request/Answer* pattern. First, a client process asks for the server's address and then starts with the *Request/Answer* pattern. The server has to register first with the broker before it becomes accessible. The broker process itself waits for a client request and compares the request with the list of servers registered. It returns the match to the client. All messages are transfered by means of the shared DS. The pattern variant chosen uses *direct communication* (see Dustar et al. [Dus03], p. 95), that is, the broker just matches client and server but the processes may coordinate directly without involvement of the broker process. Figure 4.9 shows the distributed activities by means of an UML activity diagram.

Other Names and Related Patterns. Advanced variants of this coordination pattern are also known as *Matchmaker* pattern and *Facilitator* pattern, like described by Hayden et al. [Hay98]. Further variants can be derived by adding new functions to the core purpose of the broker based on the application. For example, in trading systems, the broker adds bidding functions. When no direct coordination is permitted, a broker forwards the client's request to the service provider (similar to the *Proxy* pattern). Dustar et al. [Dus03] propose a relaxation of the roles. Here, the processes do not follow the *Request/Answer* pattern. Instead the broker uses callback functions for registered objects in case an event of interest occurs.

Implementation (Sequence of DS Primitives). Since this coordination pattern consists of three process types, three different sequences of DS primitives are derived from the UML activity diagram. The client uses a write operation (*out*()) and a consuming blocking read operation (*in*()) to retrieve the server name from the broker. Then, it uses the same sequence of operations to coordinate with the server. Equation 4.69 shows the sequence μ_{P_1} for the client P_1. The broker process reads the client's request for a service in a consuming and blocking manner (*in*()), reads the list of registered servers (*rdp*()), and writes the reference of the server (that is, its name) to the client (*out*()). Equation 4.70 describes the sequence μ_{P_2} for the broker P_2. For the server, the first space primitive in the sequence is used to register with the broker (*out*()). Then the server performs the sequence already described by the *Request/Answer* pattern. Equation 4.71 describes the space primitive sequence μ_{P_3} for the server P_3. Here, k denotes the number of times the client restarts its task, m denotes

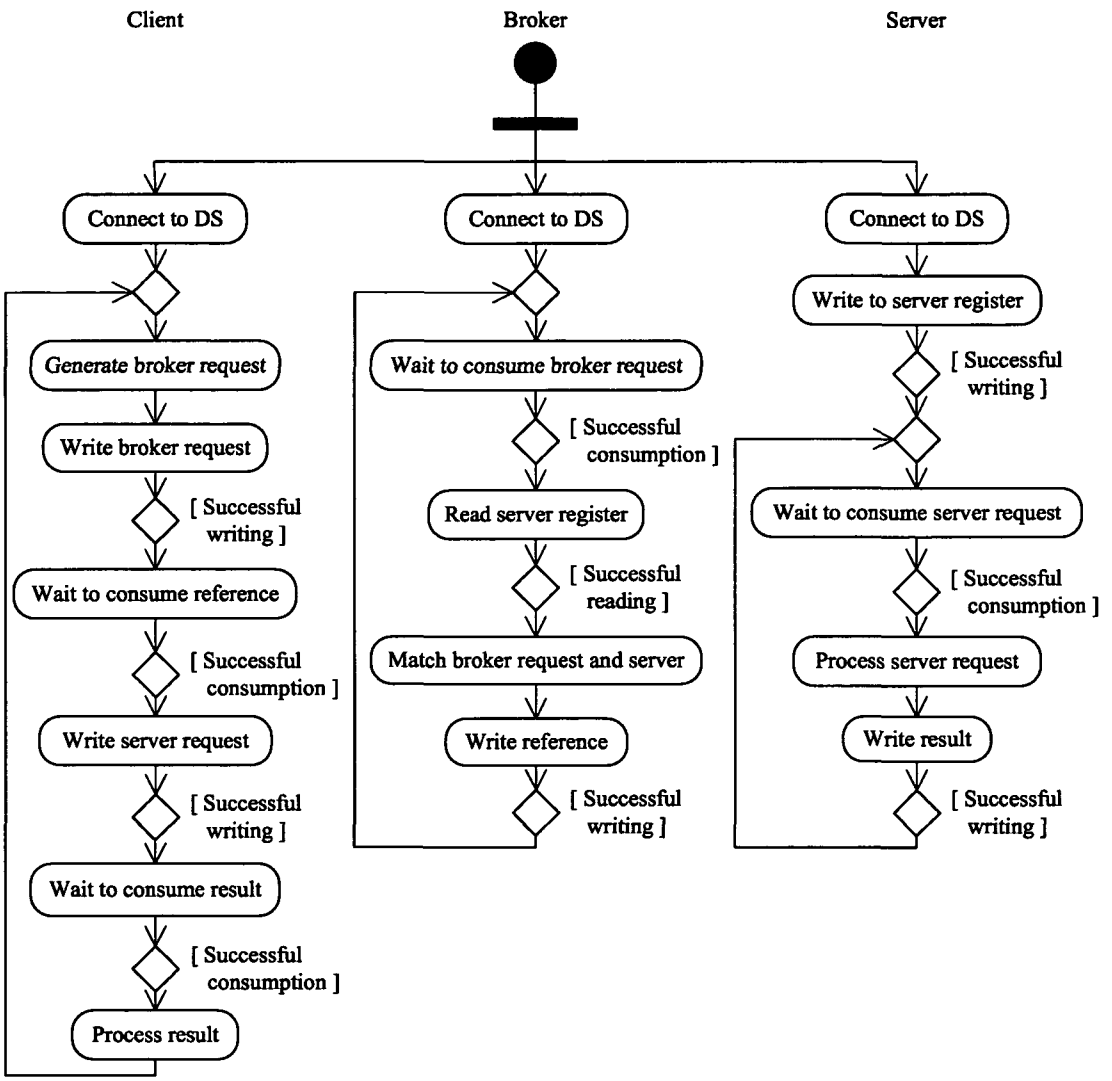


Figure 4.9: Broker

the number of times a broker restarts its task, p denotes the number of registered servers, and l denotes the number of times a server restarts its task:

$$\mu_{P_1} ::= \overbrace{out(a).in(b_{P_1}).out(c).in(b_{P_1})}^{k-times}, \text{ where } k \geq 0, \tag{4.69}$$

$$\mu_{P_2} ::= \overbrace{in(a).\overbrace{rdp(d)}^{p-times}.out(b_{P_1})}^{m-times}, \text{ where } m, p \geq 0, \text{ and} \tag{4.70}$$

$$\mu_{P_3} ::= out(d).\overbrace{in(c).out(b_{P_1})}^{l-times}, \text{ where } l \geq 0. \tag{4.71}$$

Measures. The dimension of the *Broker (Br)* pattern is:

$$n_{Br} = 3. \tag{4.72}$$

Out of the three different process types, the broker process type and the server process type are replaceable. Thus, the vectors describing the number of concurrent processes – and the number of replaceable concurrent processes – in the system are given as follows (where $c_1, c_2, c_3 > 0$):

$$CProc\_Vec_{Br} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}, \text{ and } CProc\_Vec\_Rep_{Br} = \begin{pmatrix} 0 \\ c_2 \\ c_3 \end{pmatrix}. \tag{4.73}$$

The number of concurrent processes and the number of replaceable concurrent processes are calculated as follows:

$$CProc_{Br} = c_1 + c_2 + c_3, \text{ and } CProc\_Rep_{Br} = c_2 + c_3. \tag{4.74}$$

The pattern uses three commonly known names and 1 name for each client process. Thus,

$$s_{Br} = 3 + c_1. \tag{4.75}$$

The matrix determining the number of coordination operations is given as follows:

$$COp\_Mat_{Br} = \begin{pmatrix} 2k & 2k & 0 & 0 & 0 & 0 \\ m & m & 0 & 0 & mp & 0 \\ 1+l & l & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.76}$$

Since the processes perform blocking operations, the number of times the processes loop depend upon each other. Furthermore, the number of servers registered in the server register is equal to the number of server processes in the system. Hence,

$$c_1 k = c_2 m = c_3 l, \text{ and } p = c_3. \tag{4.77}$$

The number of coordination operations and the number of blocking coordination operations are calculated as follows (based on Equation 4.73, Equation 4.76, and Equation 4.77):

$$COp_{Br} = c_3 + kc_1(8 + c_3), \text{ and } COp\_Block_{Br} = 4kc_1. \tag{4.78}$$

4.4 Discussion of Coordination Patterns

Based on the measures introduced in Section 4.2, eight coordination patterns have been described in detail in Section 4.3. These measures allow to compare the patterns in terms of referential and temporal coupling.

Table 4.3 gives an overview of the measures for each coordination pattern. Each process type is assigned to an index i (P_i). The dimension n describes the number of different process types and s denotes the number of DS names that have to be known by the participants. COp denotes the number of coordination operations, while the number of blocking coordination operations is denoted as COp_Block. Furthermore the number of concurrent processes in the system is denoted as $CProc$ and the number of replaceable concurrent processes is denoted as C_Proc_Rep.

According to the definitions of *temporally uncoupled* and *referentially uncoupled*, the eight patterns may now be classified. Thus, first the following degrees are defined respectively:

$$D_{TempUncoupled} = 1 - \frac{COp\_Block}{COp}, \text{ and} \tag{4.79}$$

$$D_{RefUncoupled} = \frac{CProc\_Rep}{CProc}. \tag{4.80}$$

A coordination pattern is temporally uncoupled, if $D_{TempUncoupled} = 1$. Otherwise it is temporally coupled. On the other hand, a coordination pattern is referentially uncoupled, if $D_{RefUncoupled} = 1$. Otherwise it is referentially coupled. Table 4.4 summarizes the coupling degree exhibited by each coordination pattern.

Now, the eight coordination patterns can be classified according to the taxonomy presented earlier. Figure 4.10 visualizes the assignment of coordination patterns to the four classes introduced by the taxonomy in Section 4.1. Furthermore, these measures allow to distinguish the degree of coupling more precisely.

Figure 4.11 shows the extended precision achieved by the measures. For each pattern, the value ranges of $D_{TempUncoupled}$ and $D_{RefUncoupled}$ as given by Table 4.4 are depicted. Table 4.5 summarizes the ranges in terms of minimum and maximum. (For a detailed deduction, see Section A in the appendix.)

To summarize, it has been shown that it is possible to classify coordination patterns according to a commonly used taxonomy which allows to assign a pattern to one of four classes in terms of referential and temporal coupling. This classification scheme has been applied to eight coordination patterns selected, which allows to chose one representative pattern per class. The influence of mobility for each class can now be studied by investigating the influence of mobility for each representative (see Chapter 7).

Coordination Pattern	n	CProc	CProc_Rep	s	COp	COp_Block
Producer/Consumer[a]	2	$c_1 + c_2$	$c_1 + c_2$	1	$c_1k + c_2m$	0
Publisher/Subscriber[b]	2	$c_1 + c_2$	$c_1 + c_2$	2	$c_2(1 + 3k)$	c_2k
Mailbox[c]	1	c_1	0	c_1	$2c_1k$	0
Master/Worker[d]	2	$c_1 + c_2$	c_2	$1 + c_1$	$4c_1k$	c_1k
Request/Answer[e]	2	$c_1 + c_2$	c_2	$1 + c_1$	$4c_1k$	$2c_1k$
Proxy[f]	3	$c_1 + 2c_2$	0	$c_1 + 2c_2$	$2k(2c_1 + c_2)$	kc_1
Consensus[g]	1	c_1	0	$3c_1$	$3c_1k(1 + c_1)$	$3c_1k(c_1 - 1)$
Broker[h]	3	$c_1 + c_2 + c_3$	$c_2 + c_3$	$3 + c_1$	$c_1k(8 + c_3) + c_3$	$4c_1k$

Table 4.3: Comparison of coordination patterns

[a] P_1: Producer, P_2: Consumer
[b] P_1: Publisher, P_2: Subscriber
[c] P_1: Peer
[d] P_1: Master, P_2: Worker
[e] P_1: Client, P_2: Server
[f] P_1: Peer, P_2: Proxy, P_3: Proxy-Peer
[g] P_1: Participant
[h] P_1: Client, P_2: Broker, P_3: Server

Coordination Pattern	$D_{TempUncoupled}$	$D_{RefUncoupled}$
Producer/Consumer	1	1
Publisher/Subscriber	$\frac{1+2k}{1+3k}$	1
Mailbox	1	0
Master/Worker	3/4	$\frac{c_2}{c_1+c_2}$
Request/Answer	1/2	$\frac{c_2}{c_1+c_2}$
Proxy	$\frac{3c_1+2c_2}{4c_1+2c_2}$	0
Consensus	$\frac{2}{c_1+1}$	0
Broker	$\frac{4c_1k+c_1c_3k+c_3}{8c_1k+c_1c_3k+c_3}$	$\frac{c_2+c_3}{c_1+c_2+c_3}$

Table 4.4: Referential and temporal coupling of coordination patterns

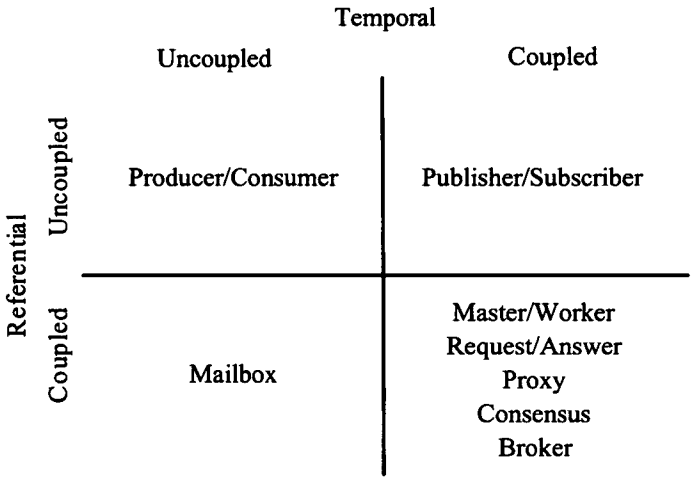


Figure 4.10: Classification of selected coordination patterns

Coordination Pattern	$D_{TempUncoupled}$		$D_{RefUncoupled}$	
	Minimum	Maximum	Minimum	Maximum
Producer/Consumer	1	1	1	1
Publisher/Subscriber	2/3	3/4	1	1
Mailbox	1	1	0	0
Master/Worker	3/4	3/4	0	1
Request/Answer	1/2	1/2	0	1
Proxy	3/4	1	0	0
Consensus	0	2/3	0	0
Broker	5/9	1	0	1

Table 4.5: Range of coupling degrees

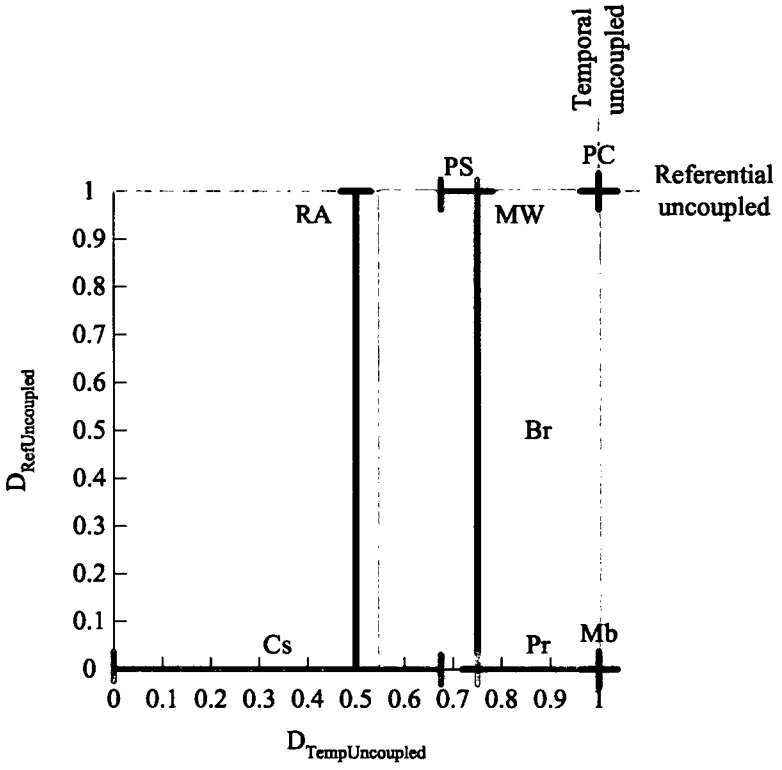


Figure 4.11: Value ranges for temporal and referential coupling degrees

Chapter 5

Mobility-Aware Space Based Computing

In mobile computing scenarios, tasks are executed on mobile, sometimes also moving devices and communicate in general via wireless networks. Adding *mobility awareness*, that is, sensing and reasoning about the roaming behavior of a user or a device, enables tasks to react and to adapt their algorithms. In case prediction is feasible, proactive adaptation of coordination tasks may help to avoid fatal problems caused by wireless network failures.

The approach presented uses mobility models in order to facilitate mobility-aware space based computing. The first section discusses the fault-hypothesis in mobile wireless systems and describes the compensating actions that can be applied in order to achieve fault tolerance. In the second section, the mobility models used for prediction are discussed. The third section introduces the *Mobility-Aware Coordination Layer (MobACL)* as part of a modular framework which combines wireless link sensing and prediction in order to alter the behavior of the shared Data Space (DS) coordination primitives.

5.1 Fault-Hypothesis in Mobile Wireless Computing

Mobile computing considers roaming behavior on different network protocol layers, like Mobile IP on network layer [Per96, Per98], or the WLAN 802.11x standards for physical and data link layer. Mobility-awareness is useful to any protocol supporting the roaming behavior of entities (for example, see the work on location augmented Mobile IP handoff from Wijngaert et al. [dW03]). In this thesis, the focus is set on indoor roaming scenarios within the same WLAN network. The fault-hypothesis is based on this focus.

5.1.1 WLAN Network Link States

Wireless links, that are, connections to a WLAN access point may be sensed in terms of the Signal to Noise Ratio (SNR) which is a means for characterizing wireless link quality (see Section 2.1). Due to experimental measures using the ORINOCO WLAN client manager (see [Age02]) and the classification proposed by the ORINOCO WLAN client manager, the WLAN 802.11b SNR value range is split into four categories as shown in Table 5.1.

EXCELLENT	MEDIUM	BAD	DISCONNECTED
$SNR > 25dB$	$15dB < SNR \leq 25dB$	$5dB < SNR \leq 15dB$	$SNR \leq 5dB$

Table 5.1: WLAN link classes

In *EXCELLENT* and *MEDIUM* state the data rates remain at 11Mbit/s, thus, it is not necessary to change the application's or middleware's behavior when changing from one of these states to the other. However, in case of observing direction of the movement it is reasonable to take a proactive action when changing from *EXCELLENT* to *MEDIUM* and thus, vice versa. When entering link state *BAD*, the data rates automatically fall back to 5.5 Mbit/s, 2.0 Mbit/s, or 1.0 Mbit/s. When no connection can be provided, the link state is set to *DISCONNECTED*.

Assuming a specific topology, locations can be mapped to the four WLAN link quality classes (also termed link quality states), like shown in Figure 5.1. As a consequence, movement from one location to another can be interpreted as a movement from one WLAN link quality state to another.

5.1.2 Definition of the Fault-Hypothesis

The approach presented assumes that wireless link degradation related to movement – which is an environmental phenomenon – leads to failures of distributed applications. Based on WLAN wireless link sensing, the changes in SNR values can be perceived and detected. Table 5.2 shows the fault hypothesis, the mapping between a possible cause of the fault (*mobility behavior*), the perception of WLAN link states (*link state*), the failures caused on application layer related to distributed computing (*coordination failure perception*), and possible compensation actions (*compensation*).

The failures considered are timing failures, which are said to be permanent (and thus, single failures only) unless the mobile entity roams in again. This behavior is similar to a repairing action commonly modeled in reliable systems. Value failures are not considered because they are hidden by underlying network

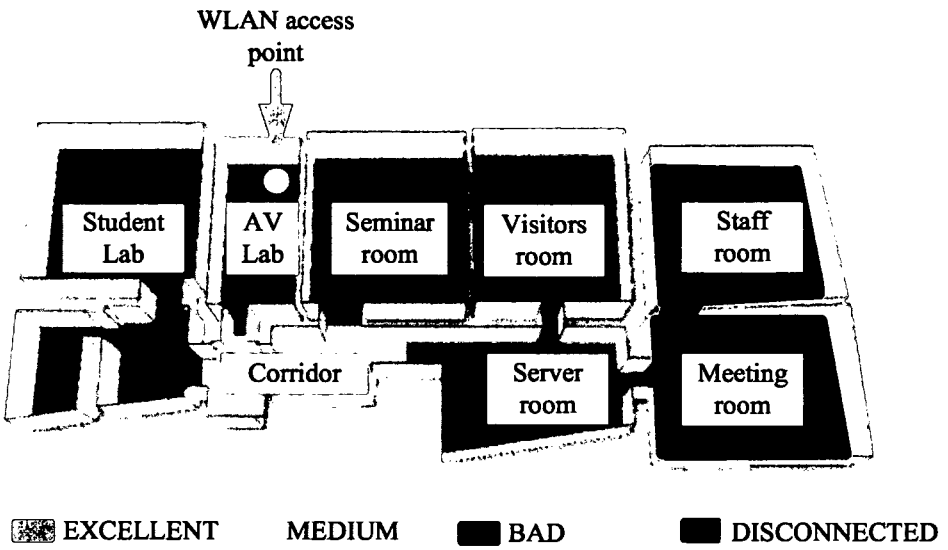


Figure 5.1: Mapping of rooms to link quality states

Mobility Behavior	Link State	Coordination Failure Perception	Compensation
Roaming in, passing through	EXCELLENT	-	-
Roaming in, passing through	MEDIUM	-	-
Roaming in, passing through	BAD	Lock timeout, access timeout	Copy, work with released locks, wait
Roaming out passing through	DISCONNECTED	Lock error, access error	Work on copy, work with released locks, wait

Table 5.2: Mobility related fault-hypothesis

protocols and the middleware layer. Furthermore, the failures are said to be consistent, that is, coordinating processes perceive the connectivity states of the others consistently. This behavior is achieved by using WLAN infrastructure mode to connect a mobile entity with others by means of only one access point.

The mobility behavior is characterized by two kinds of phenomena related to the speed of movement: *roaming in* and *passing through*. In case of a reasonable short retention period, *passing through* is assumed, otherwise *roaming in*. The

failure perception in the states *BAD* and *DISCONNECTED* on application layer is quite similar and based on the expiration of coordination related timers which may result in permanent coordination failures. The coordination related problems occur, when locks on data items are not released in time (*lock timeout, lock error*) or the access to the shared data space is not possible in time[1] (*access timeout, access error*).

In case of passing through the states *DISCONNECTED* or *BAD* – in the latter case, in addition it is assumed that the successor link state is *MEDIUM* or *EXCELLENT* – compensation actions are too expensive in terms of network and computing load. Thus, the compensation actions should be postponed (*wait*). Otherwise, in case these low quality link states are entered, only proactive actions may help to compensate connectivity related problems. In terms of the shared DS paradigm, these actions are the release of items locked during atomic operations and copy actions in order to support ongoing services which may operate on copies during periods of disconnections. When entering link state *BAD*, all data items locked should have been released in a proactive manner to avoid deadlock situations. Copy actions should be initiated which allow to prepare for connection losses that might occur. In link state *DISCONNECTED*, all shared DS coordination operations need to be finished before this state is entered. Otherwise, the operations simply cause errors.

5.1.3 Modeling Availability for Wireless Links

A mobile station roaming in a wireless network can be modeled as a Markov model of a repairable system consisting of one error-less state and one error state. This model is used to describe the overall availability of the system (see Section 2.2). Hence, the wireless link states *EXCELLENT*, *MEDIUM*, and *BAD* are here described by one state (*Connected*). A mobile device being *DISCONNECTED* is not available at the moment (*Disconnected*). Monitoring of the system starts with the first connected state, thus, also the start conditions of a repairable system are given (that is, $P_{Connected}(0) = 1$ and $P_{Disconnected}(0) = 0$). Figure 5.2 shows the application of the model for a wireless link.

For an unlimited observation period, Equation 2.13 is used to calculate the availability of the system in terms of *Mean Time To Loss (MTTL)* and *Mean Time To Connect (MTTC)*, where:[2]

$$Connection\ Loss\ Rate = \frac{1}{MTTL}, \text{ and} \tag{5.1}$$

$$Connection\ Establishment\ Rate = \frac{1}{MTTC}. \tag{5.2}$$

[1] Implicit or explicit locks are needed in order to assure consistent concurrent operations.
[2] The sum of these two mean values is termed *Mean Time Between Losses (MTBL)*.

Thus,

$$\lim_{t\to\infty} P_{Connected}(t) = \frac{MTTL}{MTTL + MTTC}. \tag{5.3}$$

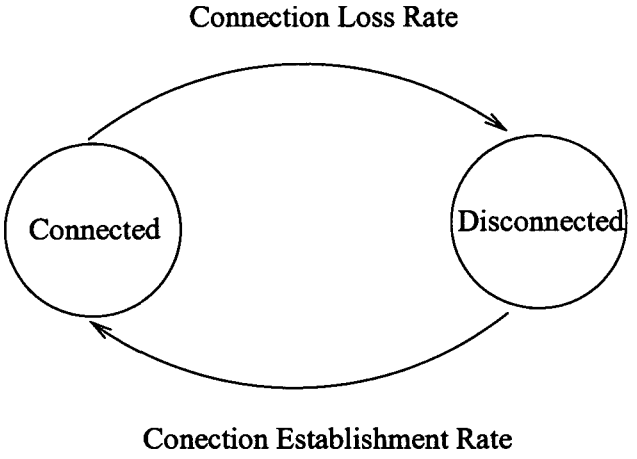


Figure 5.2: Markov model for two connectivity states

5.2 Mobility Models

Mobility models (see Section 2.1) are used to describe the roaming behavior of individuals or groups which may be used for capacity planning, proactive resource allocation, mobility aware applications like proactive parking place reservations, or proactive fault-tolerant mechanisms for middleware systems. In the context of such applications, mobility models have to provide mobility patterns at runtime. As a consequence, mobility models are most useful if they are:

- accurate in terms of agreement between mobility patterns and mobility traces,
- scalable when adding new locations and new mobile entities (either individuals or resources),
- adaptive by nature, that is, they learn from observing roaming traces, and
- inexpensive in terms of computing power and execution time needed.

Since adaptivity means spending additional computing power, mobility models are often simplified by assuming topographical models or heuristics which are used

to calculate reasonable mobility patterns before runtime. Commonly, mobility models are stored using tables which allow to compare traces against patterns easily and accurately.

A movement vector is described by its direction and speed. Thus, models describing only location sequences allow to describe direction, but not speed. The approach proposed addresses speed by modeling the retention period in states which is useful when initiating compensation actions due to wireless link state changes. Based on the notion introduced in Section 2.1, the movement history H_n is modeled according to a sequence of vectors (v_i) describing the wireless link state (s_i, which can be: *EXCELLENT*, *MEDIUM*, *BAD*, or *DISCONNECTED*) and the retention period ($r_i \geq 0$):

$$H_n = \langle v_1, ..., v_i, ..., v_n \rangle, \text{ where} \tag{5.4}$$

$$v_i = \begin{pmatrix} s_i \\ r_i \end{pmatrix}. \tag{5.5}$$

The mobility models used to describe the roaming behavior are both table-driven approaches which do not exhibit means for adaptations during runtime.[3] Thus, the models are superior to adaptive models in terms of computation cost during runtime. In comparison to random models, like the *Random Walk Mobility Model* and the *Random Waypoint Model* (see, for example, a description in Camp et al. [Cam02]), the patterns described are more applicable to the office scenario, because

- humans move based on purpose and not randomly, and, furthermore,
- the office topography usually restricts random movements.

More complex mobility models describe additional movement characteristics. Kobayashi et al. [Kob00] introduce an *Abstract Mobility State Space*, where a state of a mobile user is defined as a finite vector describing arbitrary characteristics and the interpretation of traces is based on a Markov model (*Hidden Semi-Markov Model*). The work applies this general approach to three dimensions, that are, location (finite set of locations), direction (*north,south,east,west*), and velocity (*stationary,walking,city,...*). In the work of Zaidi et al. [Zai04], such a vector describes position, velocity, and acceleration in a two-dimensional grid. By applying an autoregressive model, the next state-vector is estimated. However, the user's retention period is not modeled explicitly and can only be calculated in case the user is temporarily stationary (that is, velocity and acceleration are zero). Since the approach presented in this work aims at proactive invocation, the retention period has to be modeled.

[3] However, means for adaptations can be easily added.

Based on these aspects, two different models are developed for link state prediction as well as for retention period prediction: the *Continued Move Mobility Model (Continued Move)* and the *Smart Office Mobility Model (Smart Office)*. While the first pattern is based on general assumptions on human movement and continuous wireless link degradation, the *Smart Office Mobility Model* allows to model human movement up to a fine-grained level. It is open to additional information and scales well, because updates in granularity do not influence table entries already existing.

5.2.1 Continued Move Mobility Model

The *Continued Move* model is a second order Markov predictor assuming that the next state depends only on the last two history states [Che03c]. Since only four link states are distinguished, the Markov model is sufficiently simple for our purpose. Both the probabilities for link state transitions and retention periods can be approximated by the frequency of occurrence of state changes based on observations. For simplicity reasons, here, both variables are assumed to be independent.

The link state probability transitions described in Table 5.3 are based on reasonable assumptions, when no observed roaming frequencies are available. They describe a specific mobility pattern by taking topographical information and signal dispersion into account. Furthermore, it is assumed that a person walking with a mobile device will more likely change to neighbor locations – and thus adjacent network states (that is, states with a current SNR value corresponding to the next higher or lower interval) – than roaming like jumping randomly in a discontinuous manner. Furthermore, a moving person is expected to continue in her direction.

Giving an example in the setting depicted by Figure 5.1, in case the last two history states are *EXCELLENT-MEDIUM*, the person seems to roam away from the access point and the prediction for the next network state will most probably be BAD (1/2). In case no continuous roaming has been observed, one of the adjacent locations (link states) is assumed. For example, in case the last two history states are *DISCONNECTED-EXCELLENT* it is reasonable to derive the next link state based on the last history state only (here, *EXCELLENT*). The locations nearby the location with excellent wireless connectivity are assumed to be either *EXCELLENT* or *MEDIUM* with an equal probability of 1/2. (In case no deterministic decision is possible, a random decision is proposed).

Inspired by the scenario of roaming in an office, four different classes of retention periods are proposed. These retention periods define the duration a person stays in one link state. In detail, the classes describe the behavior of a walking person (R_1, for example, up to 30 seconds needed for roaming to the next link

$s_i - s_{i+1}$	EXC.	MEDIUM	BAD	DISC.
EXCELLENT - EXCELLENT	2/3	1/3	0	0
EXCELLENT - MEDIUM	1/4	1/4	1/2	0
EXCELLENT - BAD	0	1/3	1/3	1/3
EXCELLENT - DISCONNECTED	0	0	1/2	1/2
MEDIUM - EXCELLENT	2/3	1/3	0	0
MEDIUM - MEDIUM	1/4	1/2	1/4	0
MEDIUM - BAD	0	1/4	1/4	1/2
MEDIUM - DISCONNECTED	0	0	1/2	1/2
BAD - EXCELLENT	1/2	1/2	0	0
BAD - MEDIUM	1/2	1/4	1/4	0
BAD - BAD	0	1/4	1/2	1/4
BAD - DISCONNECTED	0	0	1/3	2/3
DISCONNECTED - EXCELLENT	1/2	1/2	0	0
DISCONNECTED - MEDIUM	1/3	1/3	1/3	0
DISCONNECTED - BAD	0	1/2	1/4	1/4
DISCONNECTED - DISCONNECTED	0	0	1/3	2/3

Table 5.3: Continued move link state transition probability matrix

state), short movement breaks (R_2, for example, up to 5 minutes), average working session (R_3, for example, up to 2 hours) and a stationary behavior lasting longer than 2 hours (R_4). In a first approach of this mobility pattern (see the first introduction of the MobACL in [Hum04a]), a transition probability matrix similar to the transition probabilities for link states has been proposed. However, it is not reasonable to assume acceleration when using just these four retention periods, thus, the model is degraded to a first order Markov model, where retention period prediction is based on the last retention period observed. A new period is only estimated when a link state change has been observed.

Table 5.4 shows the assumed transition probabilities in detail. For example, a person that stayed no longer than 30 seconds in the last observed link (that is, location) with a retention history R_1 is most probably walking and will contain walking (according to the *Continued Move* model). Thus, the retention period for the next link state is assumed to be R_1.

When long-term observations are available, the values given for this pattern in the matrices may easily be exchanged by the relative frequencies observed which will more precisely characterize the pathways in a specific office.

r_i	R_1	R_2	R_3	R_4
R_1	2/3	1/3	0	0
R_2	1/4	1/2	1/4	0
R_3	0	1/4	1/2	1/4
R_4	0	0	1/3	2/3

Table 5.4: Continued move retention period transition probability matrix

5.2.2 Smart Office Mobility Model

The second model chosen is a *Smart Office* model, which utilizes personal information, like a person's calendar entries for more accurate retention period and location estimation. The model is based on a timeline metaphor describing locations (or link states, like in the presented approach) which are entered at a particular point in time. The actual retention periods are calculated as the interval from one location change to the next.

Figure 5.3 shows a timeline augmented with link state information. A link state s_i is entered at a point in time t_i and left at t_{i+1}. Thus, the retention period is calculated as $r_i = t_{i+1} - t_i$ for s_i. The vectors of the movement history are derived easily. By applying values for a specific person and situation the mobility patterns emerge based on this model. To summarize, basic characteristics of this model are its scalability and openness. On the other hand, it needs precise knowledge about a person's roaming behavior in order to allow high quality prediction.

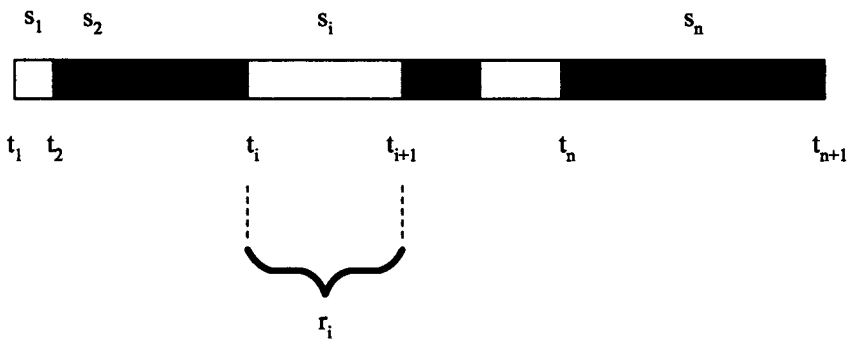


Figure 5.3: Smart office mobility model

5.3 Mobility-Aware Coordination Layer

In order to introduce mobility-awareness to mobile distributed computing on application layer, a modular software layer is proposed. This layer is termed *Mobility-Aware Coordination Layer (MobACL)* because it is meant to change the behavior of coordination primitives of an application according to the current wireless link state and the prediction of the next link state. For example, in case of being disconnected the coordination primitives will work on copies of the shared DS instead of working with the original data items. The layer should be added to a middleware based on the shared DS paradigm and is meant to execute on a mobile device capable of running both an instance of the middleware and the mobility-aware layer.

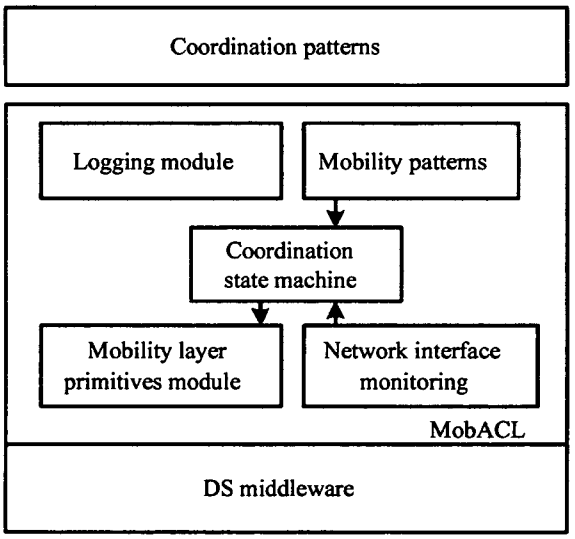


Figure 5.4: MobACL software architecture

Figure 5.4 shows the modular software architecture of the *Mobility-Aware Coordination Layer (MobACL)*. While the *coordination patterns* determine the sequence of coordination primitives used in the application – which have to be supported by the *DS middleware* – the layer alters their behavior depending on the wireless link state and prediction. The layer is made up of a *network interface monitoring* module accessing the network logs and calculating the current link state based on the modeled SNR mapping. The *mobility patterns* are used to derive a prediction for the next link state based on the history of visited link states. Additionally, they are used to derive the retention period for remaining in a link state based on the history of observed retention periods. The *co-state machine* implements the algorithm to chose the current coordination state, that

is, the state controlling the actions of the MobACL, and to execute the state transition actions. The semantics of the coordination primitives are adapted in the *mobility layer primitives module.* A *logging module* allows to log all MobACL activities of interest, like, for example, each access to the DS.

5.3.1 Co-State Machine

Primitives will be interpreted according to the *coordination state* of the application process. Four states are proposed to solve the characteristics of mobile entities and are changed on reasoning about the current link state and a prediction for the next link state and retention period periodically. Figure 5.5 shows the coordination states used and the state transitions of the MobACL. The transitions are mainly time driven but include also the event driven change in case of error. At the beginning of a transition, a decision is taken which state to enter next. Then, the methods initializing the state have to be executed. The operation *synchronize* leads to the execution of the postponed actions and is executed whenever a state transition causes a change from accessing the local space to accessing the global DS, the operation *copy* creates a copy of the needed objects in the global space, and the *release* operation releases all locks held.

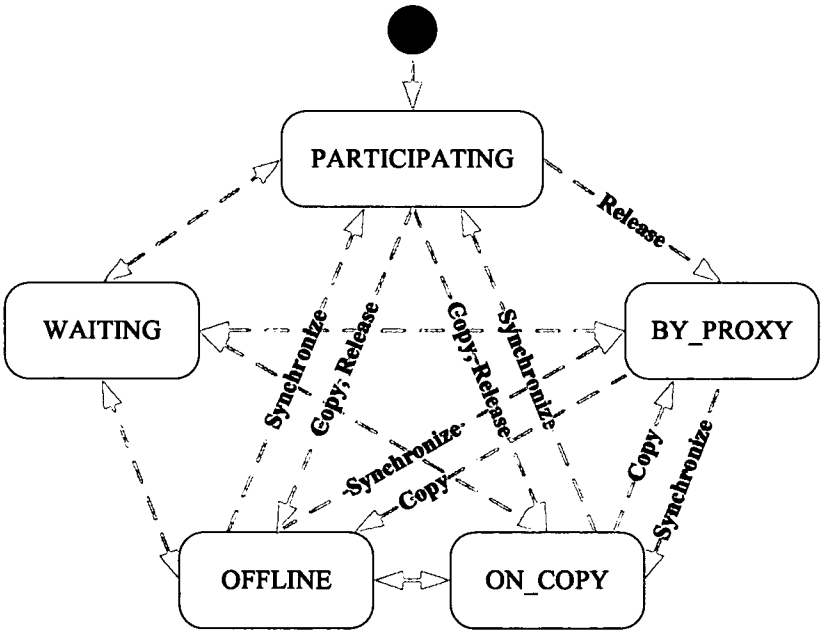


Figure 5.5: MobACL state transitions

The *PARTICIPATING* state is entered whenever the link conditions – and the prediction about the next link state – allow full participation in the global

DS. Here, the coordination primitives are applied to the global space similar to a stationary DS participant. This state is left upon link changes or because an error occurred. The latter results in changing to the *OFFLINE* state. In *OFFLINE* state, if a local copy is available, space manipulations are executed locally. In *BY_PROXY* state, the local DS is not used. Instead of using the local partition of the DS, the process on the mobile device connects to a remote DS entity and accesses the global DS via this remote site. In this state, the local site of the DS may not cause lock problems. Working *ON_COPY* allows to create a copy of the used global DS before going offline and allows to access space objects locally in order to prevent ongoing network traffic over a limited bandwidth. However, in this state triggered by a refreshing timeout, synchronization and copy of the local space are carried out in order to support working with the most recent global DS view possible. Finally, *WAITING* is used to prevent time consuming transition actions in case of frequent changes between states of different link quality like very short disconnection times. In contrast, waiting is proposed. Thus, for example, trashing is avoided in case the user resides in a location right at the border between *BAD* and *DISCONNECTED* link state. First experimental results have shown, that the *WAITING* state is only beneficial in case a short retention in the *DISCONNECTED* link state is predicted. Thus, the algorithm proposed in Hummel [Hum04a] has been adapted. After a waiting period, the previous coordination state is entered and the next transition takes place from the previous coordination state to the next coordination state. In order to prevent long process stalls the *WAITING* state has to be left after a defined timeout.

5.3.2 Calculating the Current Coordination State

The decisions in the state transition diagram (Figure 5.5) depend on the location of the entity (here, mapped to link states) and the remaining retention in the current location. Figure 5.6 shows the algorithm used to decide whether the current link state or the estimation of the next link state is the crucial factor for determining the current coordination state. In case the current retention period is still greater than a given threshold (*D_RELAX*), the *current_link_state* is the deciding factor (*controlling_link_state*). Otherwise, the *next_link_state* is the determining factor.

However, in case the user is expected to stay only a short period of time in the current link state *DISCONNECTED*, the *waiting_flag* is set. Since the *WAITING* state is a special case that can only be entered for a limited period of time, it is further checked whether the variable *waiting_time_expired* is yet false. The second special case occurs in current link state *BAD*, whenever the next link state is supposed to be better in terms of link quality (that is, not *DISCONNECTED*). In these cases, it is not recommended to execute synchronization operations while still being weakly connected, that is, operating with reduced DTR.

```
if current_duration > D_RELAX then
  controlling_link_state = current_link_state
else
  controlling_link_state = next_link_state
  if current_link_state = DISCONNECTED then
    if next_link_state ≠ DISCONNECTED then
      if not waiting_time_expired then
        waiting_flag = true
      else
        controlling_link_state = current_link_state
      end if
    end if
  end if
  if current_link_state = BAD then
    if next_link_state ≠ DISCONNECTED then
      controlling_link_state = current_link_state
    end if
  end if
end if
```

Figure 5.6: Determining the controlling link state and the waiting condition

Based on the *controlling_link_state*, now the coordination state can be derived. In case the *waiting_flag* is set, the algorithm decides to change the *current_co_state* to *WAITING* state. Otherwise, the *current_co_state* is changed determined by the *controlling_link_state*. Figure 5.7 shows how the coordination state *current_co_state* is derived.

```
if waiting_flag then
  current_co_state=WAITING
else
  if controlling_link_state = EXCELLENT then
    current_co_state = PARTICIPATING
  end if
  if controlling_link_state = MEDIUM then
    current_co_state = BY_PROXY
  end if
  if controlling_link_state = BAD then
    current_co_state = ON_COPY
  end if
  if controlling_link_state = DISCONNECTED then
    current_co_state = OFFLINE
  end if
end if
```

Figure 5.7: Determining the current coordination state

Chapter 6

Application

The concept of the *Mobility-Aware Coordination Layer (MobACL)* is implemented mainly by means of the shared object based middleware CORSO [Küh01, TEC04][1] and the Java&Co Application Programming Interface (API). The software is purely written in the Java[2] programming language and uses a WLAN 802.11b client manager from Orinoco[3] for WLAN sensing and logging.

Section 6.1 describes the main characteristics of the technologies used. Design decisions and the software overview are discussed in Section 6.2. Section 6.3 describes the interfaces between the MobACL and coordination pattern implementations. It discusses the mapping between the sequences of abstract shared *Data Space (DS)* primitives from Section 2.3 and Java&Co API operations. Furthermore, this chapter applies the implementation approach to four example reference coordination patterns introduced in Chapter 4.3. Finally, implementation alternatives and the configuration of the MobACL are discussed.

6.1 Systems, Tools, and Programming Languages Used

MobACL is based on three software systems and tools as shown in Figure 6.1. The prototype layer is purely written in the Java programming language (Java 2 Standard Edition (J2SE)) and thus, accesses CORSO via the Java&Co API. This API is a Java library providing access to the CORSO for Java applications. Finally, MobACL makes use of the Orinoco WLAN client manager [Luc02], which provides WLAN link quality measures. Since MobACL addresses mobile devices

[1] CORSO originates from research carried out under the guidance of Prof. Eva Kühn at the Vienna University of Technology.

[2] http://java.sun.com/

[3] http://www.agere.com/

capable of running the CORSO middleware, notebooks are the target platforms considered.

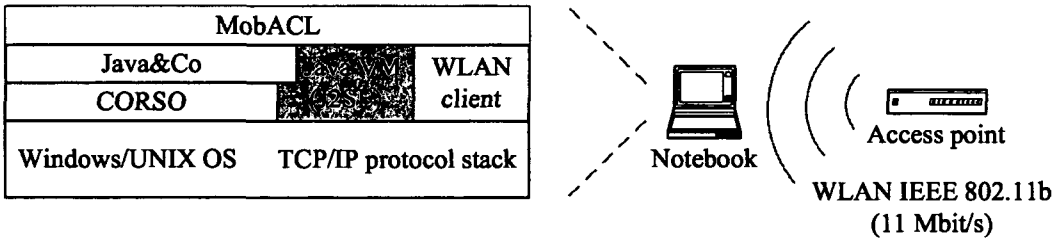


Figure 6.1: MobACL software architecture overview

6.1.1 Shared Object Space CORSO

The implementation of the proposed approach uses the space based middleware CORSO version 3.3 [TEC04], which provides advanced means for fault-tolerant space based coordination in a distributed system [Küh01, Küh02]. CORSO implements a virtual shared memory as shown in Figure 6.2. CORSO runs a *Coordination Kernel (CoKe)* on each site participating (in Figure 6.2, CORSO-site1 and CORSO-site 2). Hence, each CORSO site implements a part of the distributed shared space. The CoKes coordinate concurrent data access, consistent data replication, and persistent data storage. Processes (in Figure 6.2, P_1–P_6) use CORSO APIs in order to communicate with remote processes and to start processes on remote sites by means of CORSO.

Mobile devices can make use of CORSO in two different ways. In case they are able to execute a CORSO runtime environment, that is, a CoKe executing on a CORSO site, they may participate in the shared CORSO space like stationary computers. On the other hand, if mobile devices do not run a local CORSO site – because, for example, they do not exhibit enough memory, processing power, or software support – they may connect to a remote site via UDP and sockets. Figure 6.2 depicts a process P_6 which is, for example, executing on a PDA with Java support. MobACL supports only the first type of scenarios because it addresses full participation of mobile nodes for space based distributed computing.

Main services provided by CORSO are advanced means for reading, deleting, and writing of arbitrary data structures by means of *communication objects*. In particular, MobACL uses the following means provided by CORSO and supported by the Java&Co API:

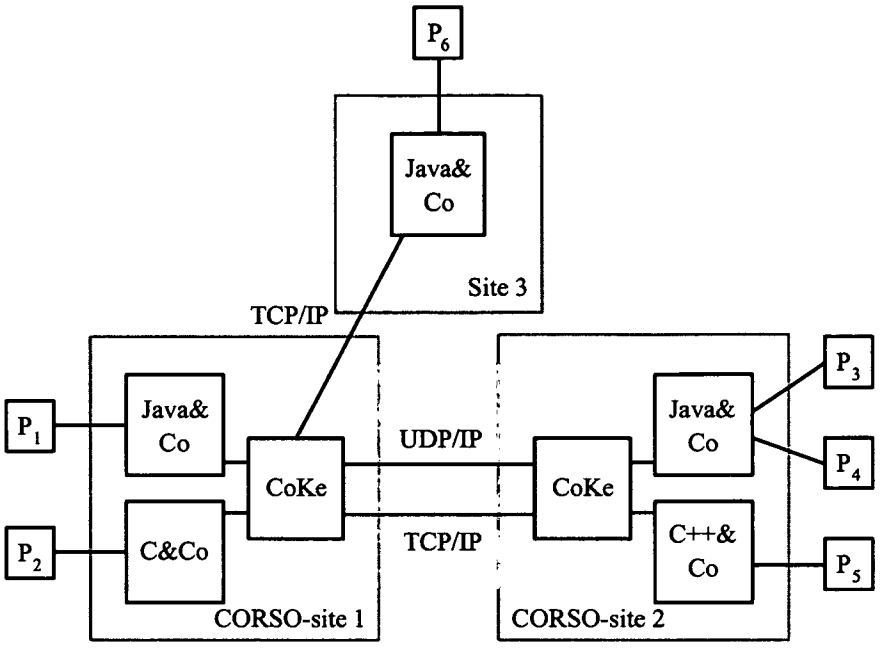


Figure 6.2: CORSO architecture

Communication Objects. Communication objects are used in order to communicate between concurrent processes and to synchronize between such processes. Each communication object is uniquely identified via an *object identifier (OID)*. There are two kinds of communication objects: variable and constant communication objects. A variable communication object can be updated arbitrary times while a constant communication object can be written to only once. Logical timestamps count the number of times a value has been written to a variable communication object. The following operations are defined for communication objects:

- a/synchronous read,
- write,
- test (which is a cheaper operation than an asynchronous read),
- notification (which can be used to wait until a new value is written to one communication object out of a group of communication objects of interest).

The OID of a communication object has to be known in order to access this object. OIDs can be passed by means of process arguments, for example, an argument passed to a new thread of a process. Furthermore, OIDs can be contained in another communication object shared by the process or they

can be retrieved via a name, in case the communication object is a *named object.*

Named Objects. If a name is assigned to a communication object, the OID can be retrieved by querying CORSO. Since a global naming strategy is not intended by CORSO, names are locally managed at CORSO sites. As a consequence, both the site and the name have to be known in order to retrieve the OID of a named object.

Shareable Data Structures. CORSO allows to use basic data types, like Integer (32-bit integer value), *Single Precision Floating Point* (32-bit floating point value), and String (0 terminated character sequence). Furthermore, it is possible to define more complex data structures for shared data exchange. Each shared data structure is automatically converted into an Interface Definition Language (IDL), termed *Native IDL.* Thus, CORSO supports heterogeneity in terms of programming languages, operating systems, and hardware architectures.

Transactions. Transaction support is an advanced mechanism provided by CORSO to assure consistency of multiple operations. CORSO transactions are based on the *Flex Transaction Model,* which relaxes the isolation property of a transaction (see Bukhres et al. [Buk93]). Thus, CORSO avoids locking of communication objects which is beneficial in distributed systems. All other ACID properties, that are, atomicity, consistency, isolation, and durability, are guaranteed by the model. Furthermore, CORSO transactions provide backtracking and compensation activities [Küh01, TEC04].

CORSO supports fault tolerance and fast access to communication objects by means of replication techniques (part of the *distribution strategies* [TEC04]). Currently, CORSO only supports the *passive redundancy with a deep object tree (PR-deep)* strategy. This strategy is a reliable strategy based on the concept of one primary copy of a communication object and multiple secondary copies, which are updated according to the CORSO strategy selected. *Eager* propagation assures that an updated value is propagated instantaneously to all sites holding a copy. Using *lazy* propagation, a copy is updated in case a process accesses this copy.

The replicas build a tree structure. Whenever a process requires write access (or read of a communication object in a transactional manner), the primary copy migrates to the CORSO site requesting it and the object tree changes, if possible. Thus, the protocol assures mutually exclusive access to replicated communication objects. In case a process does not alter a communication object, a secondary copy is accessed. Figure 6.3 shows an example object tree under change. First, the primary copy migrates from CORSO site 1 to CORSO site 2. Second, the primary copy migrates from CORSO site 2 to CORSO site 3. The CORSO site

holding the primary copy is the root of the object tree. After a migration only the root of the object tree is changed, that is, the relation between the previous root and the new root, while the other relationships of the tree's nodes (that are, parents and children) are not changed.

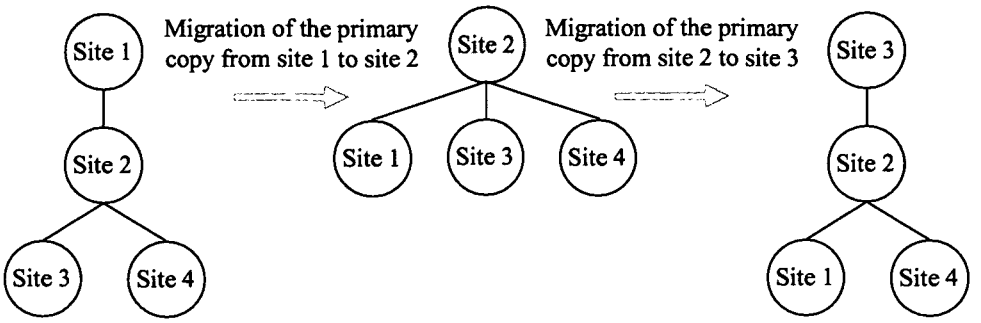


Figure 6.3: Example CORSO communication object tree

For mobile devices which may frequently disconnect, it is highly recommended to refrain from holding any primary copy. If the holder of a primary copy disconnects, other processes requesting access to the primary copy are blocked. However, in case mobile devices are full participants in the CORSO space, this problem may occur. One possible solution is implemented by MobACL via the *release()* operation introduced in Section 5.3.1.

6.1.2 The Java Programming Language

Since the first launch of the *Java Development Kit (JDK)* in 1995, the Java programming language has been successfully used in various application areas. The Java language supports advanced means for distributed computing, like, Java Remote Method Invocation (RMI), applets, and component based Web-services by means of Java beans. Thus, it is most frequently used for network computing applications. Java exhibits the following main advantages:

Platform Independence. Since Java programs are translated into a machine independent byte code, it is possible to execute them on arbitrary platforms without re-compilation (*write once, run everywhere*). This argument is still valid, although it has been weakened mainly by the introduction of the *Java 2 Micro Edition (J2ME)* which supports not all packages supported by the standard edition (J2SE). Furthermore, the *Java Native Interface (JNI)* threatens platform independence because it allows to include code written in other languages which are usually compiler based.

Object Orientation (OO). Java implements a strong object oriented concept. Every new class defined is derived from one main superclass termed *object*. This concept is only weakened by the language mechanisms introduced by *interfaces*, which allow to build a secondary line of inheritance and by built-in integral types, like *int* or *boolean*.

Concept of Multi-threading. Java allows to handle threads by means of core language primitives. It is possible to define, start, and synchronize threads without additional libraries or packages.

Security. Since Java has been proposed as a network language, remote access has been thought of from the very beginning. It is possible to control access of remote programs via *policy* definitions. Policies are used in order to grant and prohibit access to resources.

In addition to the use of core language functions, some advanced Java features are used by MobACL. Since they are important for the modular concept of the layer, they should be described briefly:

Interfaces. In the inheritance hierarchy it might be necessary to declare a superclass without any data members, but only operations. In that case, Java interfaces can be used. Similar to classes, interfaces can be used to build inheritance hierarchies (note, that multiple-inheritance is permitted for interfaces, but not for classes). Classes may *implement* interfaces, which means that they have to implement all operations specified by the interface. This language mechanism allows to use operations of a class which is not known before runtime. During runtime, a class implementing such an interface can be bound to its interface definition (*late binding*).

Packages. Multiple related compilation units (that are, units consisting of one main class and, eventually, several embedded classes), may be composed by means of packages. Thus, packages can be used to modularize software systems. Furthermore, packages define a common scope for the data types declared within the package. Each compilation unit is assigned to exactly one package (which has to be specified in the source code).

Introspection. Classes inherit some advanced language functions from the root of the Java object hierarchy, which provide information about the class itself. This mechanism is termed *introspection*. For example, a reference to the class itself can be requested by executing *getClass()*. Furthermore, the name of a class, its declared operations, fields, and constructors might be requested. Since Java supports the instantiation of an object by knowing the name in String format, this mechanism can be used to implement advanced polymorphic source code.

6.1.3 ORINOCO WLAN Client Manager

For WLAN connectivity, ORINOCO classic gold PC cards are used. These cards are compliant to the IEEE 802.11b standard and enable network connections up to 11 Mbit/s while operating in the 2.4 GHz unlicensed frequency band. The card supports *dynamic rate shifting* in case the link quality decreases. In this case, the maximum data rate is decreased to 5.5 Mbit/s, 2 Mbit/s, or 1Mbit/s automatically.

The tool used for WLAN sensing is the ORINOCO 802.11b WLAN Client Manager described in the user manual by Lucent technologies [Luc02]. This software provides both means for visualizing the WLAN link state and a logging function. Figure 6.4 shows the screen shots of link states differentiated by the client manager. In case the client manager is not indicating excellent or good connection, the user guide recommends to roam closer to the Access Point (AP). However, it is not documented how the link states are calculated. Thus, the link states introduced in Section 5.1 are related to the link states of the WLAN client manager, but do not exactly correspond to these states. In our terms, as a rule of thumb, *EXCELLENT* is used for *Excellent*, *MEDIUM* is used for *Good*, *BAD* combines *Marginal* and *Poor*, and *DISCONNECTED* is used for *No* connection. (*Peer2Peer* connections are not used by MobACL.)

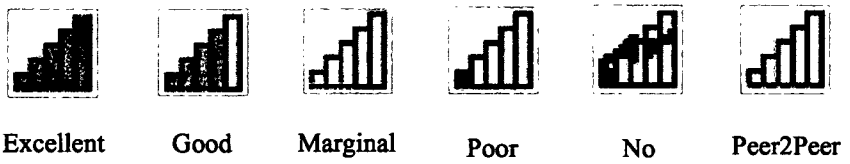


Figure 6.4: ORINOCO WLAN client manager link states

The logging function of the ORINOCO client manager allows to trace important properties of the current link quality periodically and store it into a log file. However, the periods configurable via the client manager may not fall below one second. Minimum, maximum, and average values are stated for the following properties:

- local/remote signal to noise ratio,
- local/remote signal level,
- local/remote noise level, and
- lost/received test messages.

The log file is read periodically by MobACL in order to calculate the link quality. Here, the average local signal to noise ratio (SNR) is evaluated.

6.2 Software Architecture of the MobACL

MobACL provides a pattern independent software layer which is open to new mobility patterns. The design of the layer and the software structure developed assure convenient usability of the layer.

6.2.1 Software Design Decisions

The design of MobACL aims at providing an easy to use software framework for WLAN network sensing, for prediction based on mobility patterns (mobility-awareness), and for the alteration of coordination behavior. In addition to design issues like ease of maintenance, correctness, and understandability, the design addresses the following main challenges in particular:

- openness,
- ease of extensibility, and
- traceability.

In order to apply MobACL to different distributed applications, openness to various coordination scenarios is a main issue. Thus, MobACL defines a set of core operations which can be used by distributed applications instead of the corresponding original CORSO Java&Co operations. Based on the current coordination state, CORSO operations are executed either on the global space or on local copies.

Two mobility patterns derived from the mobility models introduced in Section 5.2 are included in the MobACL. Since these patterns are only a subset of possible mobility patterns, it is important to be open for additional mobility patterns. MobACL assures extensibility by means of an interface definition for mobility patterns. For example, each mobility pattern has to implement an operation returning a prediction of the next link state for MobACL.

Finally, in order to evaluate the effect of mobility and thus, varying WLAN link quality states, it should be possible to store traces of the experiment persistently. MobACL addresses this issue by adding a logging module which produces log files consisting of timestamped traces describing state changes, coordination operations, and the success of such operations.

MobACL does not aim at implementing all possible CORSO Java&Co API operations, but only the operations needed for performing the basic shared DS coordination operations. However, the modular structure of the layer allows to add new API function easily. This characteristic is also beneficial when the layer has to be adapted to new CORSO releases.

6.2.2 OO Software Structure

The software structure of the MobACL includes packages and classes used for modularization purpose. Furthermore, interfaces are used to assure openness. By adding reflection, these interfaces are also used to enable late binding of objects. Figure 6.5 shows the UML component diagram of MobACL consisting of three packages: *Coordination*, *Core*, and *Mobility*. Furthermore, this figure depicts how these packages are decomposed into main classes and lists the most important operations. Classes and interfaces depending on CORSO are gray colored, the other classes are independent of the underlying middleware.[4]

The *Coordination* package includes only the interface definition for coordination patterns. Any coordination pattern which uses MobACL has to implement the operations defined in the *CoPattern* interface. While the operations *read()*, *write()*, and *delete()* are used to access single communication objects, the operations *synchronize()*, *release()*, and *copy()* are used to implement the transitions between coordination state changes (see Section 5.3) which consist of multiple accesses to communication objects. The operation *init()* is used to initialize the CORSO data structures. All these operations are executed on coordination pattern specific data structures. Thus, they can only be implemented by a specific coordination pattern class.

The *Core* package of MobACL consists of a class responsible for deriving the current link state (*LinkState*). In detail, the SNR is calculated by evaluating the last ten values of the average local SNR generated by the WLAN client manager. After assigning the corresponding logical link state, the majority of these ten logical link states is assigned to the current link state. The class *StateMachine* is responsible for calculating the coordination state based on the current link state and the prediction of the next link state proposed by the mobility pattern selected. Furthermore, this package consists of a class implementing different coordination operations depending on the coordination state (*CoOperation*). Depending on the coordination state, this class differentiates between working on the global space or a local copy and, thus, hides the coordination state from coordination patterns. Two operations are depicted in addition to the operations defined by the *CoPattern* interface. The operation *getNamedObject()* retrieves a

[4]For simplicity reasons, negligible class relationships and negligible class members are left out. Operations listed in an interface are not listed again in the class implementing the interface.

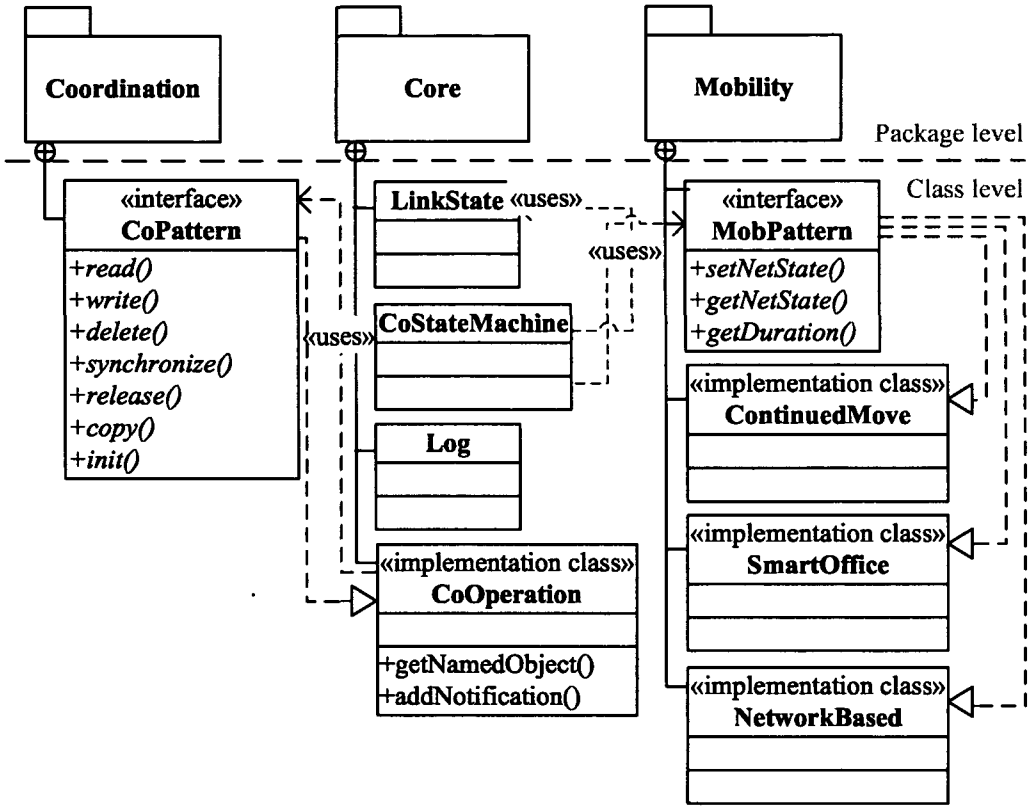


Figure 6.5: MobACL OO software structure

VAR or CONST OID from CORSO identified by a name and a CORSO site (that is, IP address). The operation *addNotification()* allows to set up a notification on a specific CORSO communication object. Furthermore, the class *CoOperation* implements operations to establish connections and to close connections to CORSO sites. Furthermore, transactions are supported by the *CoOperation* class. Finally, the package consists of a class responsible for generating traces by means of log files (*Log*).

The *Mobility* package consists of an interface definition for mobility patterns (*MobPattern*), two implementations of selected mobility patterns (*ContinuedMove* and *SmartOffice*) and one implementation of a pattern that is not able to predict the next link state for comparison purpose. This pattern depends on the current link state only (*NetworkBased*). In case a new mobility pattern should be applied to MobACL, a new implementation class of the *MobilityPattern* interface has to be created.

6.3 Coordination Pattern Implementation

Since CORSO is based on the shared DS paradigm, data structures are the central element for distributed communication. In addition to basic CORSO data structures like strings or integer values, applications often need to define more complex data structures. While access to the shared space is possible and efficient, pattern participants access the shared space (here, termed *global space*). On the other hand, MobACL may decide to work on copies instead (here, termed *local site*). When working with the local site, activities performed have to be tracked and postponed.

All patterns use a set of activity flags for each communication object to keep track of postponed operation. The common semantics of the flags used for the four patterns are defined as follows:

NOTHING. This flag denotes that no activity has to be performed on the communication object.

DELETE. This flag indicates that a communication object has to be deleted from the global space, in case the item still exists. Otherwise, it is ignored.

CREATE. This flag denotes a creation request of a new communication object.

WRITE. This flag denotes a write attempt to an existing communication object. In case the object does not exist any more, the semantics of this case has to be defined by each coordination pattern.

6.3.1 Mapping DS Operation to Java&Co API Operations

CORSO is not fully compliant with the original DS approach, but extends this approach and alters some characteristics. The differences are mainly caused by different identification approaches. In contrast to the basic shared DS model which uses templates to identify data tuples, CORSO uses communication object identifiers (OIDs). However, it is possible to use named CORSO objects. Consequently, if only one name for DS primitives should be used, a structure for addressing multiple CORSO OIDs via this name is required.

The implementation of the coordination patterns uses a single link list structure for this purpose as shown in Figure 6.6 for one data item. The first communication object in the list is accessible by its name. Communication objects may be added to the list and consumed or read from the list. In order to access communication objects in the list, they have to be searched first. This may cause several read operations until the corresponding list element is found. In cases

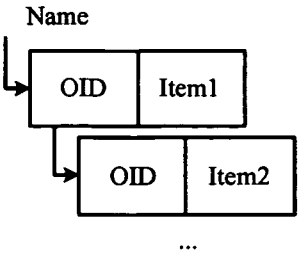


Figure 6.6: CORSO list data structure

where a linear search is not acceptable ($O(n)$), alternative data structures should be used, like for example hash tables ($O(log\ n)$, $O(1)$).

With the simple list structure, the semantics implemented by the basic DS primitives are mapped as follows:

Write Operation (*out*()). In contrast to the original DS approach, producing new items in CORSO requires a creation of a CORSO communication object followed by a write operation. Because of this semantical alteration of the basic write operation, CORSO allows to update existing data items (in CORSO terms, VAR communication objects) using one operation only.

Consume Operations (*in*(), *inp*()). The consumption of a data item (*in*()) can be implemented by applying a Java&Co sequence of a read and a delete operation. Timeout values can be used to implement either blocking or non-blocking semantics to the read operation. Since the MobACL has to reflect on the wireless link state periodically, only virtual blocking is applied, that is, the operation is invoked until it succeeds.

Read Operations (*rd*(), *rdp*()). The semantics of reading of communication objects is similar to the semantics of the DS primitives. Here again, virtual blocking is used to implement blocking operations.

Process Creation Operation (*eval*()) . The Java&Co process classes provide advanced means to create and start processes. However, these operations are not used in the coordination pattern implementations.

Additionally, CORSO supports *notifications* in case communication objects are altered. Since this mechanism simplifies event processing, it is used in addition to the basic shared DS based operations described above.

In case the atomicity of multiple concurrent operations have to be assured, such operations are guarded by CORSO transactions. A transaction commits

successfully in case all operations can be performed without interference of another concurrent process. In CORSO terms, a transaction commits only in case the process can get access to the primary copy of all concerned communication objects. In case an error occurs, transactions are aborted. Finally, when using timeout values greater than zero and different from waiting infinitely (*INFINITE_TIMEOUT*), a transaction may also be aborted in case the timeout expires.

6.3.2 Coordination Pattern Software Structure

Figure 6.7 depicts the UML component diagram of a coordination pattern. Each coordination pattern consists of a description of the data structures used (class *DataStructure*). In addition, for working on copies, the class *CopyDataStructure* is used which extends the original data structure by the former discussed activity flags. Since the operations performed on the data structures are dependent on the data structures and on the organization of the data items, these operations have to be implemented by each coordination pattern (implementation class *CoPatternImpl*).

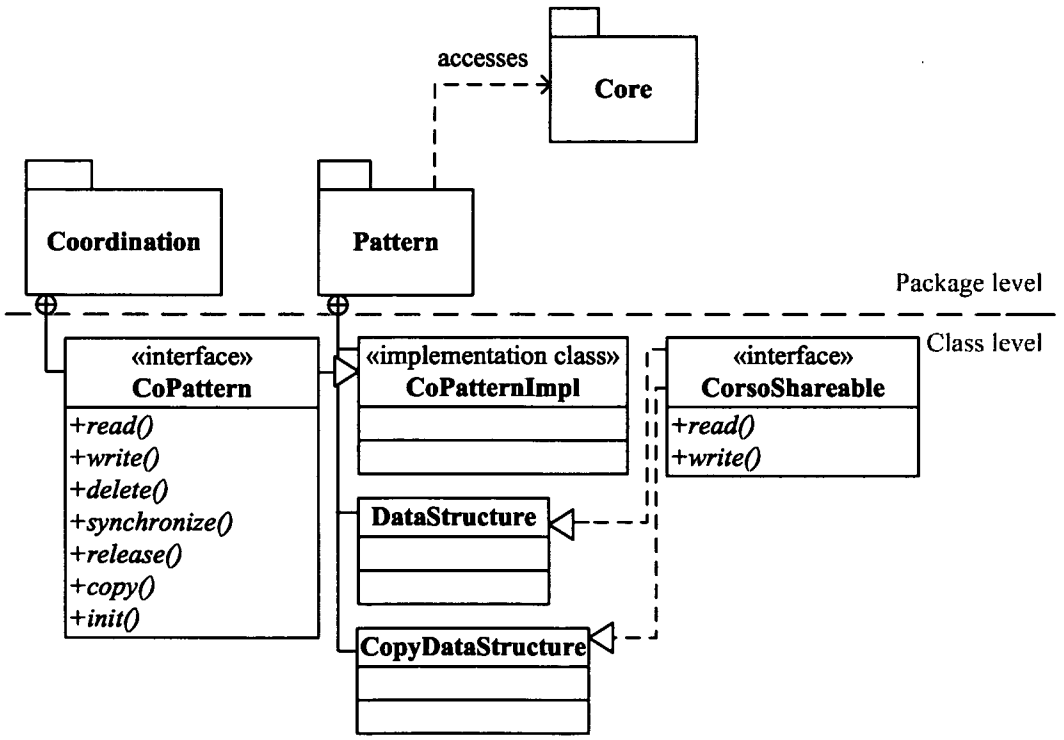


Figure 6.7: OO software structure of coordination patterns

The *copy()* operation replicates the content of every communication object of interest. During the *release()* operation, the process tries to release all primary copies currently held. Finally, during the *synchronization()* operation, the activity flags of all communication objects on the local site are processed and the corresponding postponed activity is executed. Consequently, after a successful completion of this phase, all communication objects processed are synchronized.

6.3.3 Data Structures

The remaining parts of this section introduce the data structures used for four coordination pattern representatives. The selection is based on the classification introduced in Chapter 4.

Producer/Consumer

Figure 6.8 shows how CORSO communication objects are managed as a single linked list both in the global space and the local space (for a detailed discussion see [TEC04]). In the global space, producers create new CORSO communication objects identified by OIDs, link these new communication objects with the previous ones at the end of the list (*End of stream*) and write the data to the new communication object (for example, *Item 1*). The start of the list is accessible via a name (in CORSO terms, *named variable communication object*). Consumers read communication objects and delete them, starting at the beginning of the list (*Start of stream*). Since concurrence conflicts have to be avoided, CORSO transactions are used to assure consistency. When copying data items from the global space to the local site, data items are copied only for the consumer process type because only this process type can use produced items.

The local copy data structures consists of additional *Activity flags* indicating the action to perform on the original communication object in the global space. The producer uses the flags *CREATE* and *WRITE* in order to postpone the production of items. The consumer uses the flag *DELETE* in order to postpone the deletion of communication items read. Since in this simple variant it cannot be assured that a data item is processed by one process only, at least once semantics is exhibited by this pattern.

Publisher/Subscriber

The implementation of the *Publisher/Subscriber* pattern makes use of CORSO notifications. A group of subscribers are notified whenever a publisher updates a communication object. Then, these subscribers read the communication object. In contrast to the push mode variant described in Section 4.3, the implementation

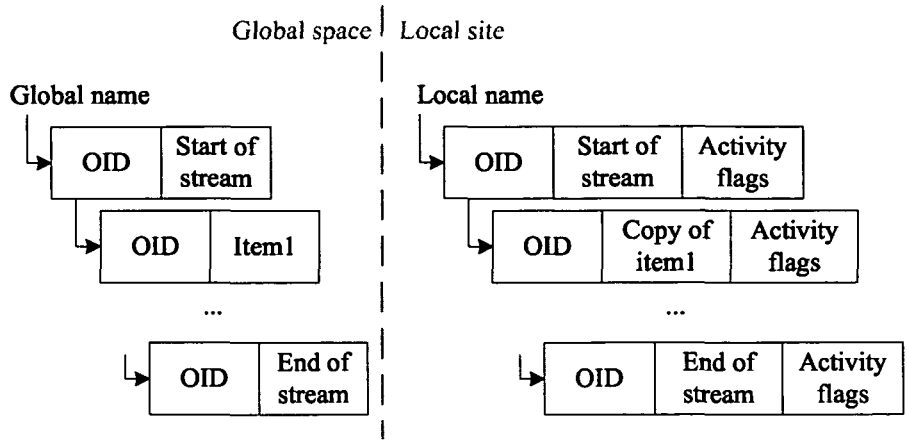


Figure 6.8: Producer/Consumer data structures

is based on a pull mode, that is, subscribers have to retrieve the data performing an additional step after the notification has been received. Since no acknowledgment mechanism assures that all subscribers have read the update before a new value is published, this pattern exhibits at most once semantics.

Figure 6.9 shows the data structures used. Publishers write to one single named communication object (*Item*). Each subscriber adds a notification request bound to this communication object. In case a state change causes a copy operation of the global space to the local site, only the *Item* is copied. Notifications are not copied because no other process may cause the firing of a notification bound on a communication object of the local site. For the *Item*, the *WRITE* activity flag is used to track updates on the copy.

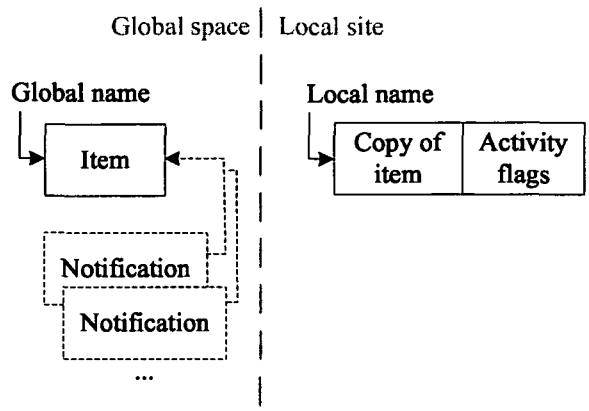


Figure 6.9: Publisher/Subscriber data structures

Mailbox

The data structures used by the *Mailbox* coordination pattern are similar to the *Producer/Consumer* data structures. Here, a global and a local mailbox is maintained for every known peer. Each peer accesses its own mailbox as a consumer and the foreign mailboxes as a producer. Thus, the activity flags used are again *DELETE* for postponing a consumption and, for producing a new item, *CREATE* and *WRITE*.

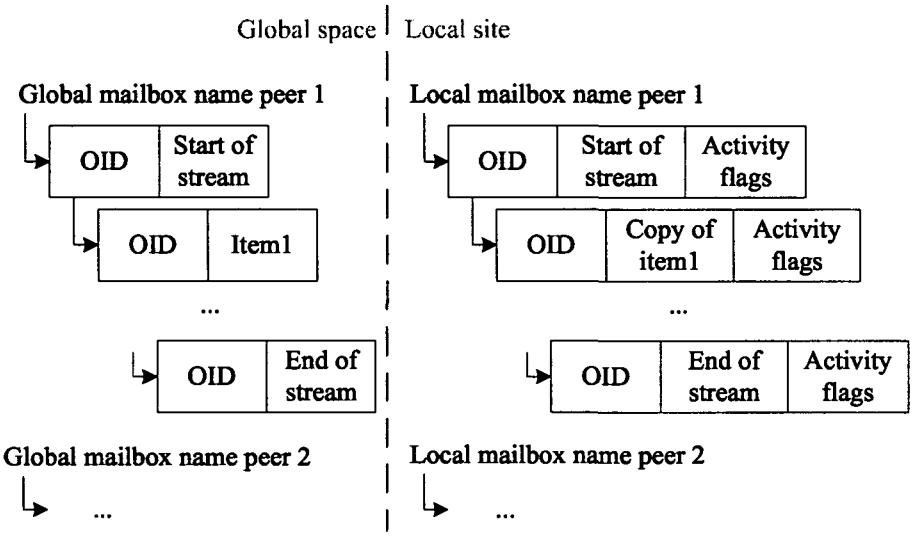


Figure 6.10: Mailbox data structures

During the copy operation, only the messages (communication objects) stored in the peer's own mailbox are copied. Local copies of the mailboxes of the other processes are only used to store new messages which are synchronized during a coordination state change. Thus, the pattern's copy algorithm is optimized.

Request/Answer

The data structures used by the *Request/Answer* coordination pattern consists of linked communication objects. Each communication object consists of a reference to the next request, a request value and a reference to the communication object the answer should be written to (*C_OID*). Server processes consume a request and write the answer to the communication object identified by a C_OID. A Client process produces a request which includes the creation of the communication object used for the answer. Then, this process adds the request at the end

of the service list and waits until an answer has been written to the answer communication object (C_OID). Figure 6.11 shows the data structures used.

When operating on copies, clients may produce a new request (indicated by the flags *CREATE, WRITE*) but waiting for the answer has to be postponed until the request has been synchronized with the global space. Additionally, it makes no sense to copy answer communication objects. Instead, the server alters the copy request entry by substituting the request value with the answer value. The activity flag evaluates to *DELETE*. During the synchronization phase, the request is deleted after the answer has been written to the answer communication object.

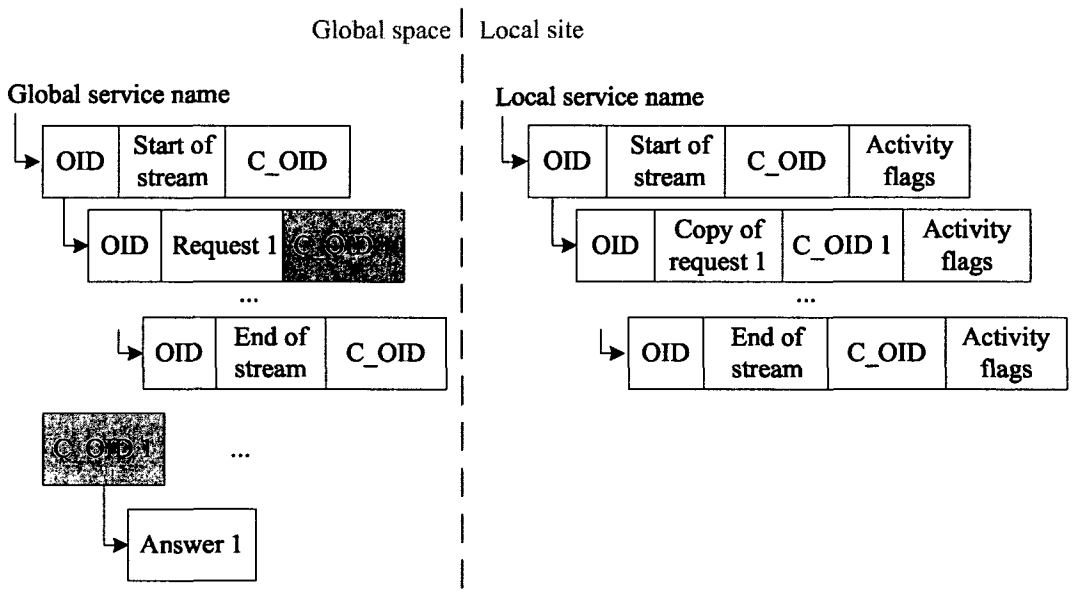


Figure 6.11: Request/Answer data structures

6.3.4 Configuration

The framework presented can be configured via two packages in order to provide different timing and workload behavior. First, in the *Core* package of MobACL, the timing behavior of CORSO operations and the state machine may be changed by altering the following parameters (all are defined in units of seconds):

OP_TIMEOUT. All Java&Co operations defined in the *Core* package of MobACL use this unique timeout value. In case an operation cannot complete within *OP_TIMEOUT* seconds, a timer expires and interrupts the operation. This mechanism prevents MobACL from blocking infinitely. This

timing constraint effects the outcome of an operation significantly. For example, in case the timeout value is too small, no operation may be able to succeed.

RELAXATION_THRESHOLD. This value is used to define the minimum estimated retention period necessary for a new link state. MobACL uses this threshold in order to prevent state change overload in case a state is left again soon. Following the algorithm presented in Section 5.3, in such cases MobACL might change into a *WAITING* state.

WAITING_TIME. This value defines the period MobACL may stay in the *WAITING* state.

LINK_POLLING_TIMEOUT. This value determines how often MobACL recalculates the current coordination state by means of WLAN logs and mobility pattern based prediction.

Second, the coordination patterns may be parameterized. In case coordination patterns are compared with one another, it is recommended to choose similar values:

PROCESSING_TIME. This number is a means for defining the time a participant spends for processing during one round.

ITEMS_PER_ROUND. This number denotes the number of communication objects that should be accessed during one execution round.

MAX_ITEMS. Each process is allowed to produce new communication objects up to this threshold. Once this threshold is reached, the processes refrain from producing any more items.

6.4 Discussion

MobACL consists of a modular structure where the most classes are independent of the underlying space based middleware CORSO. Mainly the coordination patterns and their supporting core operations depend on CORSO and the Java&Co API. MobACL is extensible in terms of coordination patterns and mobility patterns. In detail, new coordination patterns may be introduced by implementing the coordination pattern interface. Similarly, new mobility patterns just have to implement the mobility pattern interface.

The example representative coordination patterns are implemented based on a few design decisions about data structuring. A single linked list is proposed as the basic structure for communication objects that are addressable via one

name only. However, the list data structure exhibits some major drawbacks in terms of scalability. In case many participants try to get exclusive write access concurrently, this structure will slow down throughput. Similarly, it can be argued that for a huge number of linked items, linear search is a bad decision. Thus, the structure is well suited for the purpose of this work, but will need some adaptations if used for a huge number of participants.

Alternatively, in order to reduce the number of concurrent accesses to the same data items, these items can be virtually distributed by a hash table, for example, consisting of multiple different item lists. Another alternative can be derived by using a tree-structure. Furthermore, the number of transactional read operations can be reduced, for example by allowing to access also the tail of the list directly.

Chapter 7

Evaluation

As has been stated previously, that the *Mobility-Aware Coordination Layer (MobACL)* provides means for proactive coordination support using movement prediction. Additionally, this chapter proves and discusses the benefits and limits of MobACL quantitatively. Experimental evaluation is carried out using four distributed coordination patterns and two different mobility patterns.

The mobility patterns selected exhibit different potentials for next link state prediction and for next retention period prediction (see Section 5.2). Hence, differences between those predictors can be observed. The *Continued Move Mobility Pattern* is derived by applying a Markov model to the indoor wireless network scenario, while the *Smart Office Mobility Pattern* has been selected as an optimal mobility predictor based on accurate user schedule information.

On the other hand, depending on the coordination pattern selected, mobility causes different effects and MobACL achieves different results. Each coordination pattern selected exhibits different coupling characteristics in terms of time and reference and, thus, represents one class according to the taxonomy for distributed computing presented in Chapter 4. The comparison of the results derived enable a discussion about the potential for mobility-awareness. In addition to MobACL based achievements, replication of processes is investigated as a second approach.

The experiments are carried out in two different ways. First, experiments are conducted by physical roaming applied to a reference coordination pattern. Second, experimental evaluation is carried out for all coordination patterns by means of simulation. This approach allows to compare the results of the simulation with the original system for a reference pattern and, thus, proves that the simulation is based on a reasonable model and, further, allows proper parametrization of the simulation. Additionally, the original failure rate of space operations is observed and logged while roaming physically.

Section 7.1 introduces the measures used for evaluation, while Section 7.2 describes and argues the concept and setup of the experiment. Section 7.3 in-

cludes a description of the MobACL configuration and the simulation environment used. Section 7.4 and Section 7.5 describe the experimental results derived. In Section 7.5, first temporal coupling is addressed by investigating the reference coordination patterns while moving. Additionally, referential coupling is addressed by investigating the potential for redundant process execution for each coordination pattern. Finally, Section 7.6 summarizes the results and discusses the achievements and limits of the MobACL and process replication.

7.1 Evaluation Measures

In order to study the influence of mobility and of mobility-awareness, measures are necessary which allow to compare the observed effects. The measures are defined based on the following questions:

- How many distributed activities have been finished successfully?
- How many overhead operations have been necessary to achieve proactive behavior based on mobility-awareness?
- How many erroneous operations have been carried out?

The first question addresses the distributed computing quality achievable in terms of coordination pattern throughput. Going one step further, the second question addresses the overhead caused and the third question addresses operation failures. The following measures are defined for the corresponding three categories:

Effectiveness. In order to evaluate effectiveness or coordination pattern throughput, the *Number of Items* processed by a coordination scenario is measured over time.

Efficiency. Two measures are used to describe the operations caused by MobACL. First the number of global space operations *(Number of Operations)* describes the general global space access behavior. Second, the *Overhead Rate* is studied. This measure is based on the Number of Operations and the *Number of Overhead Operations*, that is, the number of operations caused by synchronize, release, and copy activities. The Overhead Rate is denoted as follows:

$$Overhead\ Rate = \frac{Number\ of\ Overhead\ Operations}{Number\ of\ Operations}. \tag{7.1}$$

Failure Rate. This measure is used to compare erroneous global operations caused by degraded wireless link quality. The *Failure Rate* is measured

as the relation of the number of erroneous global space operations (*Number of Erroneous Operations*) and the Number of Operations as follows:

$$Failure\ Rate = \frac{Number\ of\ Erroneous\ Operations}{Number\ of\ Operations}. \tag{7.2}$$

The measures are applied to the experiments as follows:

Experiments Based on Physical Roaming. These experiments are used to evaluate the MobACL under operation in a WLAN hotspot area. All four measures defined are applied to these cases.

Simulation Based Experiments Addressing Temporal Coupling. For the experiments based on simulation, only three measures are applicable, that are, the *Number of Items*, the *Number of Operations*, and the *Overhead Rate*. The failure rate is only discussed for the physical roaming experiment because the failure types observed are similar for all patterns and no new insights can be derived by repetition.

Simulation Based Experiments Addressing Referential Coupling. In addition to the experiments conducted for evaluating temporal coupling, these experiments should provide additional insights by including participant replication. The experiments are compared with one another in terms of the *Number of Items*.

7.2 Experimental Setup

The experimental evaluation approach is based on investigating a coordination scenario under different mobility cases as depicted by Figure 7.1. A *scenario* consists of a specific coordination pattern and a specification of pattern participants which includes the number of participants and the place of execution, that is, the host executing the process. Each scenario is applied to two different cases, first, to a *stationary* case in which coordination activities are observed without roaming. The second case considered assumes *mobile* behavior which is based on a unique timing and movement schedule. The *MobACL cases* differentiated allow to observe the effects of mobility when different mobility-awareness configuration cases are applied.

7.2.1 Mobility-Awareness Configuration Cases

For the experiments, the MobACL is observed using four different mobility-awareness configuration cases. These cases are compared to the stationary behav-

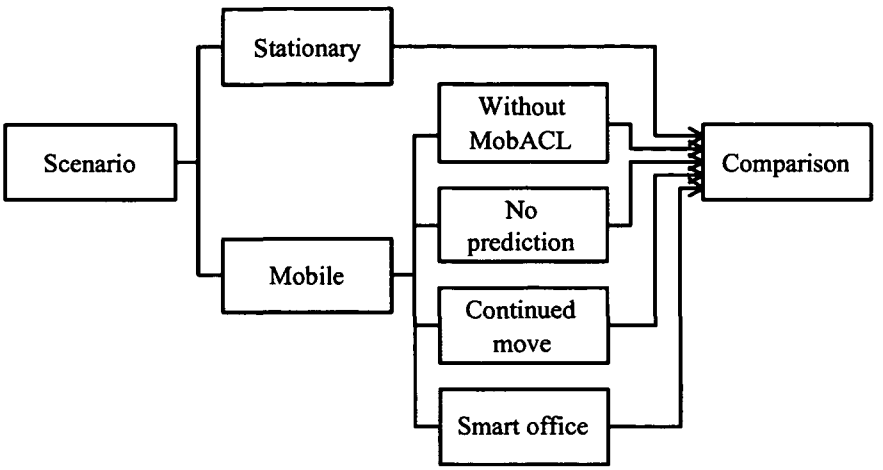


Figure 7.1: Evaluation approach

ior of the coordination pattern under study and with each other (see Figure 7.1). The mobility-awareness configuration cases are differentiated as follows:

Without MobACL. This case allows to study the behavior of a coordination pattern without using the mobility-awareness layer. Thus, mobility caused effects can be observed best when studying this case. Neither state changes, nor network sensing are considered. For simulation, this case is slightly adapted. Here, the simulated link state changes are used to invoke virtual connection losses and decreased coordination pattern throughput.

No Prediction. In this case, the MobACL is used without prediction. The current link state is analyzed and used for determining the coordination state.[1] As a consequence, on this level of mobility-awareness, only reactive behavior can be expected.

Continued Move. In contrast to the cases already mentioned, the *Continued Move Mobility Pattern* implements prediction-based state changes which allow to invoke proactive fault tolerance mechanisms. The predictors used for next link and next retention period calculation are Markov models as described in Section 5.2.1.

Smart Office. The second prediction-based mobility case is termed *Smart Office Mobility Pattern.* This mobility pattern is supposed to show the best prediction results because it allows to describe the assumed timing and

[1]This behavior is achieved by using the standard MobACL state machine and by setting the link state to the current link state.

movement schedule as precisely as possible. A timeline similar to the movement chosen for the experiment has been selected for this pattern in order to demonstrate the behavior of the state prediction algorithm in case of a perfectly matching mobility predictor. This pattern is described in detail in Section 5.2.2.

7.2.2 Timing and Movement Schedule

Since effects caused by motion should be investigated, a particular movement schedule has been chosen in order to cover all retention periods modeled by MobACL and to describe movement toward the WLAN access point and heading away from the access point. Additionally, both continuous movements and discontinuous movements are included. The link states and the retention period intervals assumed are based on the definitions introduced in Section 5.1 and correspond to the configuration of the MobACL. The retention periods have been down-sampled in order to make physical experiments possible as follows:

Very Fast Movement: retention < 30 seconds (D_1),

Fast Movement: 30 seconds $\leq$ retention < 60 seconds (D_2),

Moderate Movement: 60 seconds $\leq$ retention < 180 seconds (D_3), and

Slow Movement: retention ≥ 180 seconds (D_4).

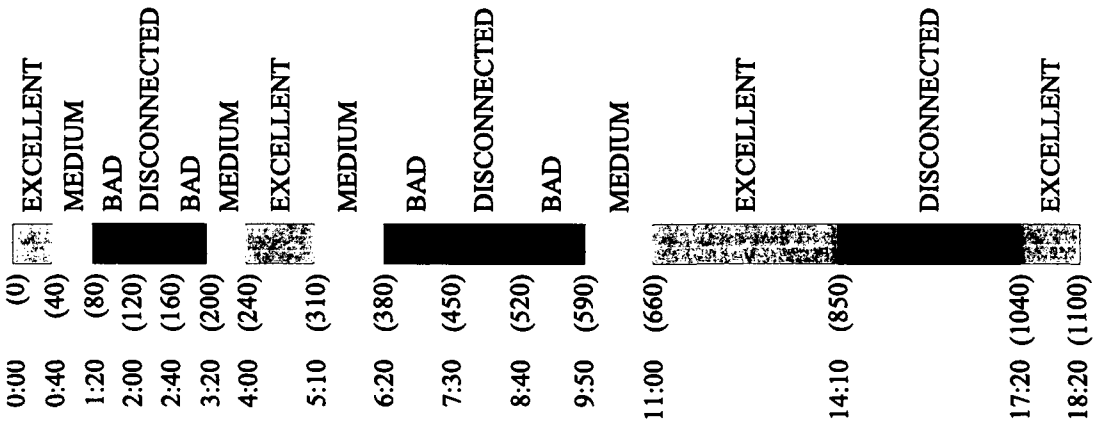


Figure 7.2: Time schedule of the experiment (format mm:ss (in seconds))

For the sake of completeness, D_1 denotes intervals which may lead to link state change misses caused by the MobACL link polling period granularity (30 seconds in the current configuration). Thus, for the experiments only the intervals D_2, D_3

and D_4 are considered. Figure 7.2 shows the relative timeline and wireless link state changes scheduled, starting with a sequence of periods which lie in interval D_2, followed by periods which lie in D_3, and finally, D_4. The last two link state changes are discontinuous changes, while the others follow the continuous model of roaming to neighboring places which correspond to neighboring link states.[2]

In terms of availability, the schedule tests the MobACL under different availability assumptions (see Section 5.1). The whole experiment may also be described in terms of *Mean Time To Loss (MTTL)* and *Mean Time To Connect (MTTC)*. Table 7.1 shows the used retention values based on the retention intervals and the corresponding values for the MTTL and MTTC.[3]

Time Schedule	Retention (Interval)	MTTL	MTTC
0 – 239 seconds	40 seconds (D2)	200 sec.	40 sec.
240 – 659 seconds	70 seconds (D3)	350 sec.	70 sec.
660 – 1100 seconds	190 seconds (D4)	190 sec.	190 sec.

Table 7.1: Retention periods covered by the experiment time schedule

In total numbers, the movement schedule proposed consist of a period of 590 seconds dwelling under good network conditions (*EXCELLENT*, *MEDIUM*), 210 seconds dwelling under *BAD* link quality, and 300 seconds staying in *DISCONNECTED* state.

7.3 System Setup

This section describes the hardware and software setup used for experimental evaluation. The configuration of the MobACL is described in detail as well as the simulation model.

7.3.1 Hardware and Software Setup

All experiments are carried out by using the same two notebooks, WLAN 802.11b (11 Mbit/s) connections, and fast Ethernet connections (100 Mbit/s). One notebook is used as the mobile host with lower capabilities (processor speed: 560 MHz, 180 MB RAM) while the other is a used as a stationary host (processor

[2]Note, that prediction based on continuous movement considerations will not be able to support discontinuous link state changes.

[3]Note, that the schedule does not contain of complete cycles of intervals corresponding to MTTL and MTTC.

speed: 1.7 GHz, 1 GB RAM). All processes use the Java virtual machine.[4] and CORSO v3.3 and the Java&Co API [TEC04] Figure 7.3 depicts the hardware setup and the process types executed during the experiment. MobACL is running on both hosts. The coordination scenarios based on MobACL can be started either directly which is used for physical roaming experiments or by a simulator process.

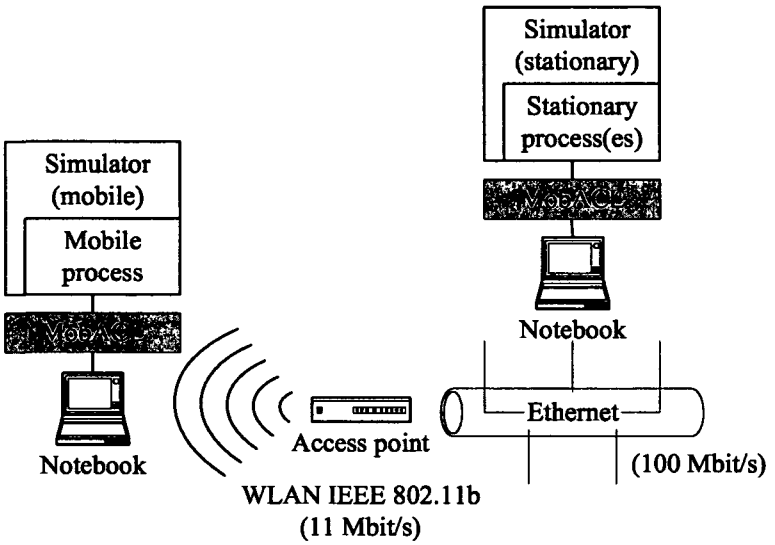


Figure 7.3: Hardware and software setup used for experimental evaluation

In case of physical roaming, the mobile node observes WLAN link state changes as provided by the Orinoco WLAN client manager. In contrast, virtual link state changes are used when working with the distributed simulator which controls the experiment sequences on both hosts.

7.3.2 Simulation

Each physical roaming experiment exhibits the potential for imprecise measures which are difficult to reproduce and to compare. Additionally, simulation enables the evaluation of a great many of different cases. Hence, a simple, but distributed simulator has been developed for synchronized execution of experiments.

The simulation model proposed allows to model the disconnection state (*DISCONNECTED*) by omitting any access to the global space. Furthermore, decreased processing for low quality link state *BAD* is simulated by a virtually decreased access success rate, that is, 25 percent of all data space accesses may

[4] http://java.sun.com/

complete successfully. In order to detect virtual link quality changes accurately, the MobACL is configured to evaluate the current link quality once per second. This configuration is based on a comparison of simulation and physical roaming experiments using a reference coordination scenario, that is, the Producer/Consumer coordination pattern applied to the *without MobACL* mobility case. Figure B.1 in the appendix shows the close approximation achieved by simulation through proper parameterization.

Since simulation results should be close to results observed by physical roaming, simulation is distributed among the same hosts used for physical roaming. Consequently, the simulator itself is distributed. Before starting an experiment, the remote simulator instances synchronize with each other by means of CORSO. Each experiment is controlled by simulating the unique movement pattern based on a source log file generated during physical roaming in the WLAN hotspot area. Figure 7.4 shows the SNR values of this source log file as they are perceived by the MobACL network module over time.

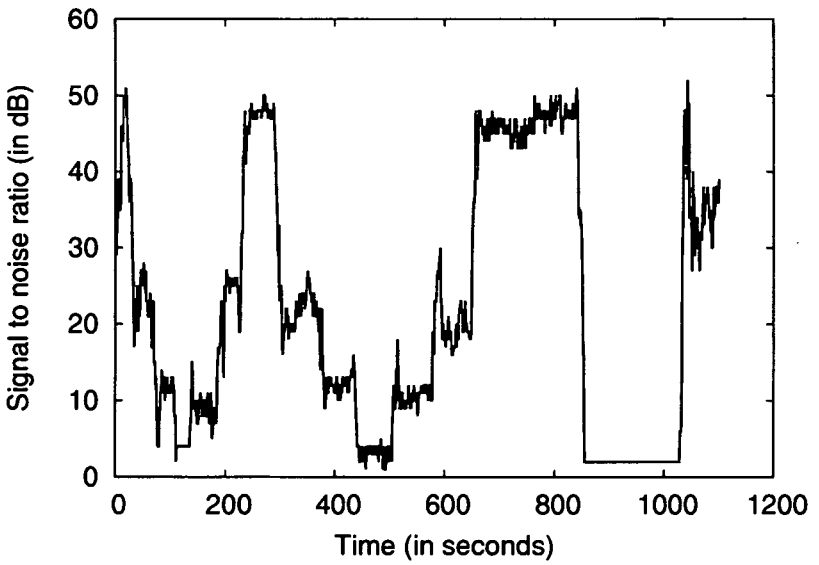


Figure 7.4: SNR measures used for all simulation-based experiments

7.3.3 MobACL Configuration

In order to achieve comparable results, the MobACL is configured similarly for each experiment.[5] Table 7.2 summarizes the configuration.

[5]For a detailed definition of the configuration parameters, refer to Section 6.3.

	Configuration Parameter	Value
MobACL Core Class	OP_TIMEOUT	30 seconds
	RELAXATION_THRESHOLD	30 seconds
	WAITING_TIME	10 seconds
	LINK_POLLING_TIMEOUT	30 seconds
CoPattern Class	PROCESSING_TIME	1 second
	ITEMS_PER_ROUND	1
	MAX_ITEMS	100

Table 7.2: Configuration used for the experiments

The timeout for CORSO operations is set to 30 seconds, as well as the relaxation threshold which determines the minimum retention period required for state change invocation. In case state changes are too costly, MobACL waits for 10 seconds and retries to calculated the coordination state again after that interval. Additionally, every 30 seconds, the MobACL calculates the new coordination state based on the current link state and a prediction based next link state, if applicable.

Each coordination pattern participant needs 1 second for processing per round and is assumed to process only 1 item per round. For each coordination pattern, the permitted limit for producing items is set to 100.

7.4 Experiments Based on Physical Roaming

The physical roaming experiment has been carried out for one coordination scenario, that is, a mobile consumer process and a stationary producer process coordinating in a Producer/Consumer manner. These experiments investigate the MobACL behavior under operation in a WLAN hotspot area and allow to discuss the failure rate caused by mobility. A systematic evaluation for each coordination class is left to the simulation approach (Section 7.5).

Effectiveness. Figure 7.5(a) depicts the cumulated items consumed during the physical roaming experiment. The effects of mobility can be observed when following the *without MobACL* curve. During disconnected times, the cumulated number of items remains constant while during times of bad link quality the increase is slowed.

The *stationary* case shows the best results achievable, that is, nearly one item per second has been consumed. For the mobile cases, the smart office mobility pattern shows the best results, followed by the reactive *no prediction* pattern which benefits from continuous link changes since MobACL

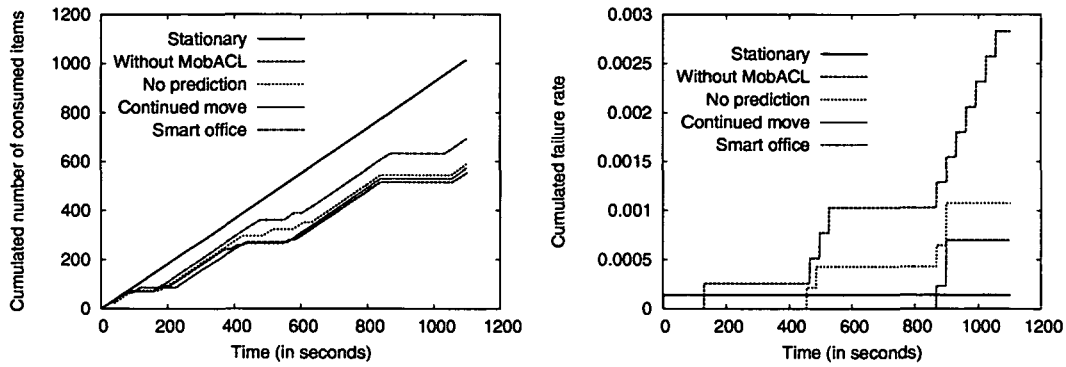


Figure 7.5: Physical roaming (Producer/Consumer): (a) cumulated number of consumed items and (b) cumulated failure rate

	Stationary	**Without MobACL**	**No Mobility**	**Continued Move**	**Smart Office**
Items consumed	1011	554	588	573	692
Failure rate	$2.83x10^{-4}$	$2.82x10^{-3}$	$1.07x10^{-3}$	$6.96x10^{-4}$	0

Table 7.3: Physical roaming: consumed items and failure rate (Producer/Consumer)

proposes working on copies when bad link conditions are observed. Thus, although the disconnected state is observed late, working on copies is possible. The continued move mobility pattern performed worse although it uses prediction. Due to the changing roaming velocity, this pattern is not able to adapt next retention period prediction accurately. Table 7.3 summarizes the total number of consumed items.

During the phase of fast movements ($0 \leq t < 240$), proactive working on copies is counterproductive. The bad effects observed are due to the few items which can be copied at that early point in time. Furthermore, when the connection is lost only for a short time, state changes are too costly. During the phase of moderate movement ($240 \leq t < 660$), the *smart office* pattern is able to support item consuming significantly better. The discontinuous change to DISCONNECTED state at $t = 850$ is only predicted by the *smart office* pattern. Here, the benefits of proactive behavior can be observed. The benefits achieved by working on copies during disconnection times can be increased when more data items are in the shared data space. Curve increases observed are also caused by transaction failures due to concurrency conflicts with the producer. In such cases, no item may be

consumed during one or several seconds.

Failure Rate. Figure 7.5(b) shows the cumulated failure rate of global space operations which is very low. For the smart office case, all global space operations have been completed successfully which is mainly caused by the accurate and proactive nature of this pattern. For the stationary case, one access error due to the concurrent creation of communication objects at the beginning of the experiment has been observed. For all other mobility cases, the curves increase corresponding to global space operations which have been carried out during disconnected periods. In contrast to the mobility cases using the MobACL which show only a few errors at the beginning of the disconnected period, the mobility case *without MobACL* generates errors lasting for the whole disconnected period since it cannot make use of a local copy. Table 7.3 summarizes the total failure rates of the experiments.

The global space operations and the overhead caused by MobACL is described in the appendix, Section C.1.

7.5 Experiments Based on Simulation

Two different types of experiments are carried out in order to study the effects of mobility and mobility-awareness in different coordination scenarios. First, the focus is set on temporal coupling exhibited by coordination patterns, second, referential coupling phenomena are discussed by replicating processes. The separation of these two cases avoids concurrency caused effects for temporal coupling based evaluation. For example, in the *Producer/Consumer* case, replicated consumers might be able to consume even more when other consumers are disconnected.

Similar to the physical roaming experiments, different mobility cases are compared with each other. Based on the classification presented in Chapter 4, the reference coordination patterns under investigation are as follows:

- Producer/Consumer,
- Publisher/Subscriber,
- Mailbox, and
- Request/Answer.

The qualitative analysis provided is based on 70 experiments which allow to compare the different coordination patterns. Since all experiments have been carried under the same system conditions, the results are expected to be representative.

Additionally, a few repeated executions carried out promise that the results do not disperse significantly.[6]

7.5.1 Addressing Temporal Coupling

For each coordination pattern, two distributed processes are involved in the experiment. In case the pattern consists of different process types, two scenarios are needed to investigate both process types on the move. Since the *Mailbox* coordination pattern exhibits only one process type, only one scenario is needed. For each pattern, the number of global operations can be found in Section C.2.

Producer/Consumer

In order to investigate the influence of mobility to this temporally and referentially uncoupled coordination pattern, two scenarios are defined. *Scenario 1* consists of a mobile consumer process and a stationary producer process, while *scenario 2* consists of a mobile producer process and a stationary consumer process.

Effectiveness. For scenario 1, similar to the results observed while roaming physically, the *smart office* mobility pattern shows the best results when moving. While during fast movement ($0 \leq t < 450$) the overhead caused by the state changes is not beneficial (using MobACL yields in slightly better results than using *without MobACL* mobility configuration), all MobACL mobility cases enable the consumption of items during the second period of disconnection. Additionally, the *smart office* mobility predictor is able to predict the discontinuous link state change at $t = 850$ accurately (see Figure 7.6(a)). In contrast to scenario 1, scenario 2 assumes a mobile producer. Since a producer does not need to copy existing items from the global space to the local site, proactive behavior is not required. Here, all three MobACL cases show quite similar results, while the reactive case (case *no prediction*) generates best results as shown in Figure 7.6(b) since this case invokes working on copies later than the other MobACL cases. Table 7.4 summarizes the total number of consumed items during the experiment for scenario 1 and 2. It is shown how mobility related link degradation decreases overall item throughput (case *without MobACL*) and how the use of MobACL compensates this effect partially.

Efficiency. The load necessary for achieving fault tolerance is summarized by Table 7.5 in terms of the overhead rate for MobACL. Due to the highly

[6] A quantitative analysis can be used to provide statistical information about the results in terms of, for example, mean values and confidence intervals. However, such analysis is beyond the scope of this thesis.

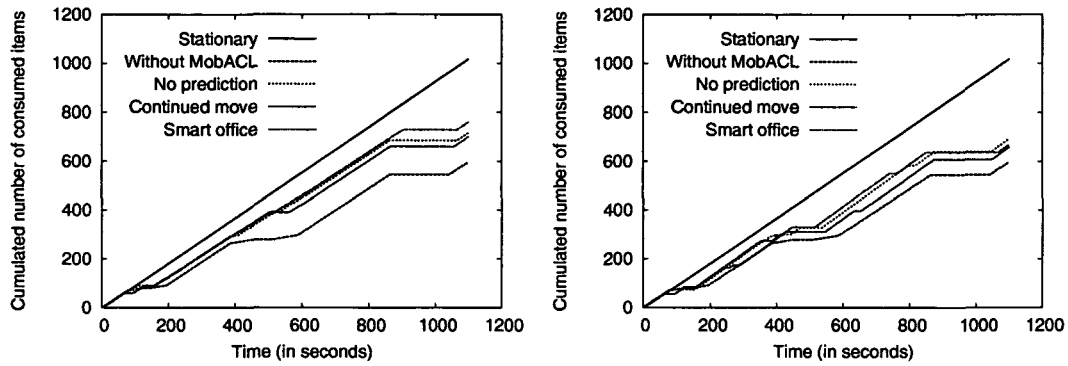


Figure 7.6: Temporal coupling (Producer/Consumer): cumulated number of consumed items for (a) scenario 1 and (b) scenario 2

	Stationary	**Without MobACL**	**No Prediction**	**Continued Move**	**Smart Office**
Scenario 1	1018	597	716	700	760
Scenario 2	1018	597	693	659	669

Table 7.4: Temporal coupling: number of consumed items (Producer/Consumer)

mobile nature of the experiments and the asynchronous operations which allow working on copies, these rates are expectedly high. Since the *smart office* mobility predictor allows the patterns to work more often on copies than the other mobility cases, it causes the highest overhead rate. The overhead operations are observed as discontinuities which appear at most state changes. In contrast to scenario 1, scenario 2 exhibits the highest increases when re-establishing operation on the global space. This effect is due to synchronization of items generated by the mobile producer. For the cases which do not make use of MobACL, no overhead operations are needed.

	Stationary	**Without MobACL**	**No Prediction**	**Continued Move**	**Smart Office**
Scenario 1	0	0	0.49	0.59	0.64
Scenario 2	0	0	0.64	0.57	0.7

Table 7.5: Temporal coupling: overhead rates (Producer/Consumer)

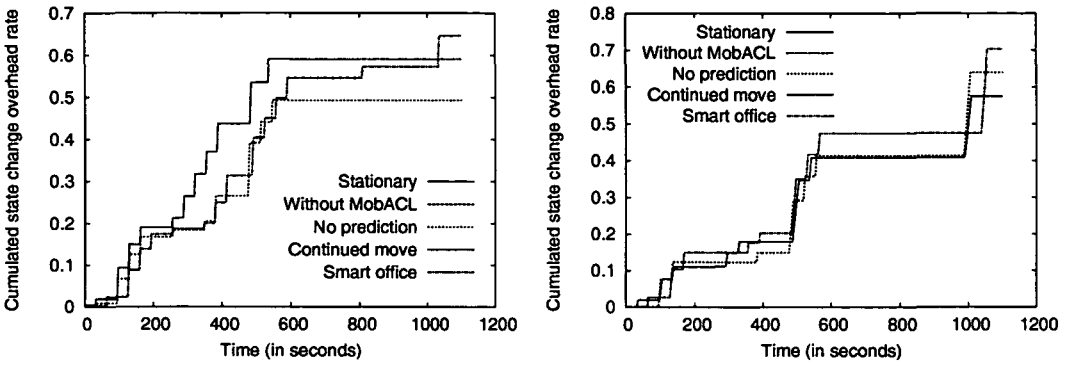


Figure 7.7: Temporal coupling (Producer/Consumer): cumulated state change overhead rates for (a) scenario 1 and (b) scenario 2

Publisher/Subscriber

Two scenarios are used for studying the temporally coupled and referentially uncoupled Publisher/Subscriber pattern. *Scenario 1* consists of a mobile publisher and a stationary subscriber, while in *scenario 2* the roles are changed to a mobile subscriber and a stationary publisher.

Effectiveness. Since the Publisher/Subscriber pattern implements synchronous communication, the processes cannot benefit from of working on copies. Proactive working on copies degrades the number of notifications even more for both scenarios as depicted in Figure 7.8(a) and Figure 7.8(b). In case of a mobile subscriber in scenario 2, the overall throughput of notifications is less than in scenario 1, which implements a mobile publisher. For scenario 2, different throughput is observed because the subscriber which executes more costly operations is executed on the low power mobile host. Table 7.6 summarizes the number of consumed updates due to notifications during the experiment.

	Stationary	**Without MobACL**	**No Prediction**	**Continued Move**	**Smart Office**
Scenario 1	974	603	535	402	453
Scenario 2	539	345	270	215	247

Table 7.6: Temporal coupling: number of received updates (Publisher/Subscriber)

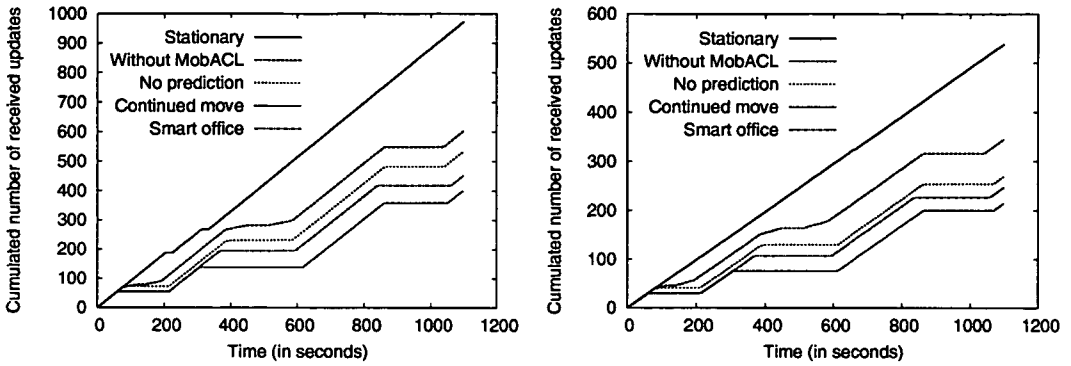


Figure 7.8: Temporal coupling (Publisher/Subscriber): cumulated number of received updates for (a) scenario 1 and (b) scenario 2

Efficiency. Only the data item, which is periodically updated by the publisher, is copied, released, and synchronized with the global space. Compared to the overhead rates calculated for the Producer/Consumer pattern, the overhead rates here are low as listed by Table 7.7. Figure 7.9(a) and Figure 7.9(b) show similar curves for all MobACL supported cases which is due to the same data item that is synchronized and copied for both participants.

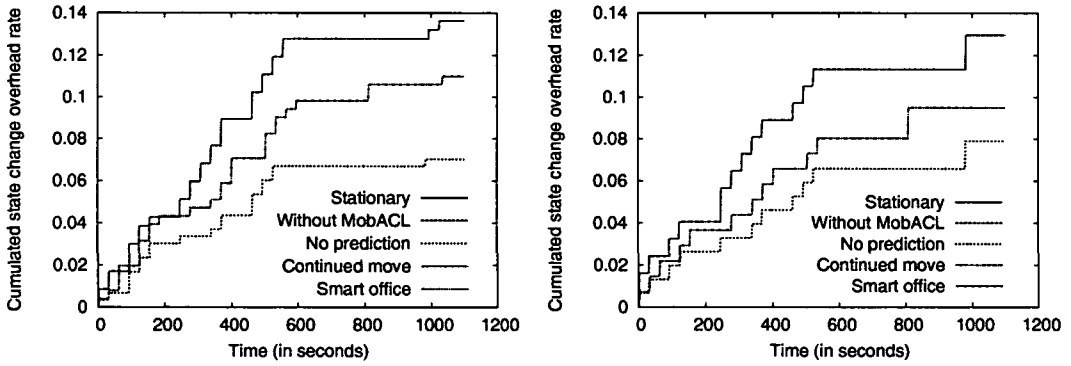


Figure 7.9: Temporal coupling (Publisher/Subscriber): cumulated state change overhead rates for (a) scenario 1 and (b) scenario 2

Mailbox

Since the Mailbox pattern consists of only one process type termed *peer*, only one scenario is investigated. One peer is executed on the mobile host while the other

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	0	0	0.07	0.14	0.11
Scenario 2	0	0	0.08	0.13	0.1

Table 7.7: Temporal coupling: state change overhead rates (Publisher/Subscriber)

one is executed on the stationary host. This coordination pattern is temporally uncoupled, but referentially coupled.

Effectiveness. Here, the messages retrieved by both peers are considered for the *Number of Items*. Among the mobile cases, the *no prediction* case slightly overtakes the *smart office* mobility pattern. Since each peer acts both as a producer and a consumer, the behavior of both process types is combined and proactive copying is not observed as beneficial as for the mobile consumer case. However, the mobile peer is able to process items stored in its mailbox while being disconnected. The *without MobACL* case shows that bad link quality causes a significant decrease of message throughput. Table 7.8 summarizes the total number of messages received by both peers during the experiment. Figure 7.10(a) shows the cumulated received messages over time. The curves show that during disconnection times, the distance between the MobACL-based curves and the *without MobACL* case is increased and after reconnecting, the *without MobACL* case changes back to high-efficient operation earlier than the other patterns which have to synchronize first. However, the second period of disconnections ($450 \leq t < 520$) shows that a significant distance remains if the disconnection interval is sufficiently large.

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Number of messages	1892	1125	1205	1176	1197
Overhead rate	0	0	0.66	0.63	0.72

Table 7.8: Temporal coupling: number of received messages and state change overhead rates (Mailbox)

Efficiency. Since a peer working on copies both consumes and produces messages, the overhead is high when assuming highly mobile behavior. Table 7.8 shows the overhead rates calculated during the experiment. For the

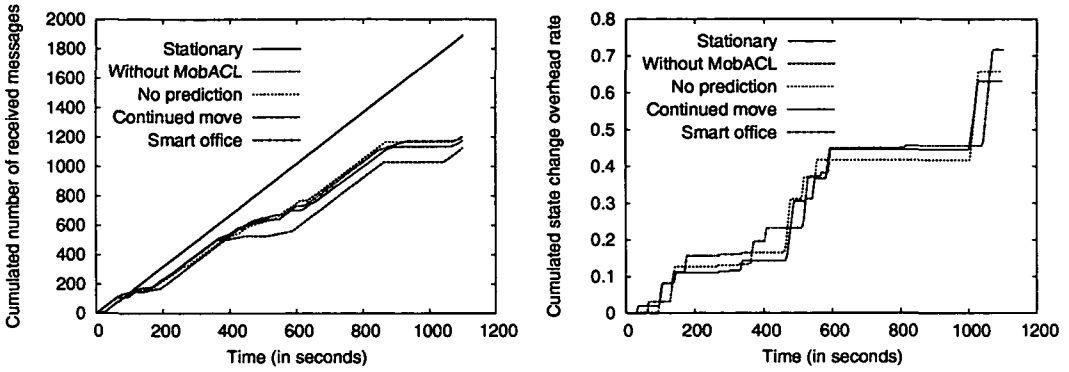


Figure 7.10: Temporal coupling (Mailbox): cumulated (a) number of received messages and (b) state change overhead rates

smart office pattern, the highest rates are calculated due to the amount of messages which have been generated and consumed during disconnection times. Figure 7.10(b) shows that at the end of a bad link quality interval the number of overhead operations executed raises significantly faster than before this link state is entered.

Request/Answer

Two scenarios are used for studying the temporally and timely coupled Request/Answer pattern. *Scenario 1* consists of a mobile client and a stationary server, while *scenario 2* consists of a mobile server and a stationary client.

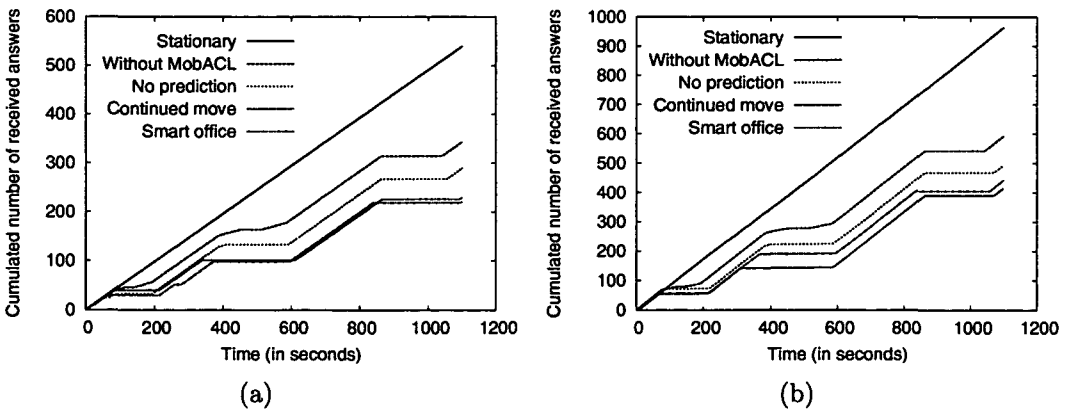


Figure 7.11: Temporal coupling (Request/Answer): cumulated number of received answers for (a) scenario 1 and (b) scenario 2

Effectiveness. As expected for this timely coupled coordination pattern, working on copies is counterproductive. Thus, the negative effects caused by mobility cannot be compensated. The more a mobility pattern suggests working on copies, the worse this pattern performs as shown by Table 7.9. Both scenarios result in similar curves over time as depicted by Figure 7.11(a) and Figure 7.11(b). The higher number of consumed answers in scenario 2 is due to different capabilities of the host computers. When the client executes on a more powerful host (stationary host), this process is able to produce more requests and, thus, receives more answers from the remote server.

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	540	343	290	229	221
Scenario 2	964	593	494	416	443

Table 7.9: Temporal coupling: number of received answers (Request/Answer)

Efficiency. State change overhead caused by the MobACL is very low because only one client request may be copied, eventually. Thus, copy, synchronize, and release operations do not cause significant overload as shown by Figure 7.12(a) for scenario 1 and Figure 7.12(b) for scenario2 and detailed by Table 7.10.

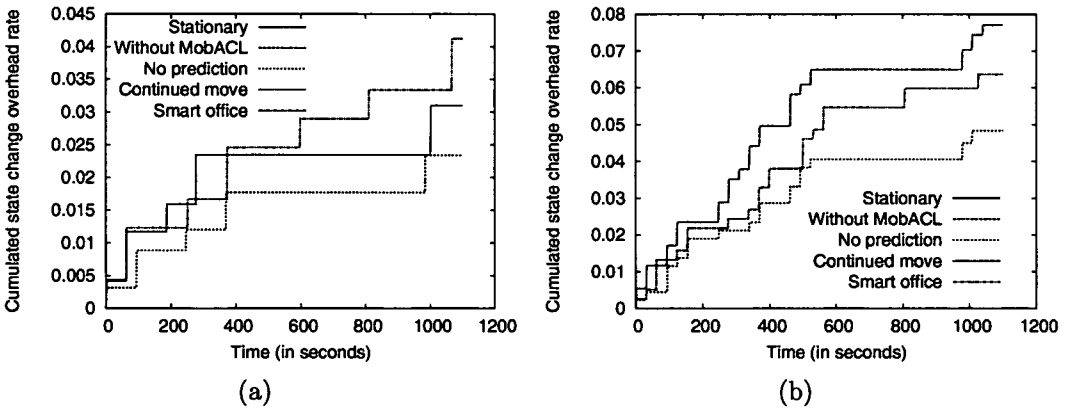


Figure 7.12: Temporal coupling (Request/Answer): cumulated state change overhead rates for (a) scenario 1 (a) and scenario 2 (b)

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	0	0	0.023	0.031	0.041
Scenario 2	0	0	0.048	0.077	0.064

Table 7.10: Temporal coupling: overhead rates (Request/Answer)

7.5.2 Addressing Referential Coupling

In order to evaluate referential coupling, the process type executing on the mobile host is replicated on the stationary host for each scenario described earlier. For the *Mailbox* coordination pattern only one process type exists (peer). Thus, two peer processes are executed on the stationary host. This setting allows to observe whether and how effective a similar process can take over in case the mobile process becomes not available. Due to the replication, concurrency caused delays will be observed at the same time. Since the experiments conducted should give new insights, only *effectiveness* is investigated.

Producer/Consumer

The scenarios for the Producer/Consumer pattern are altered by adding another consumer for *scenario 1* and by adding an additional producer for *scenario 2*. Both replicated processes execute on the stationary host.

Effectiveness. All mobility cases show better results than the stationary case in scenario 1, since all processes compete for access to one single list, which is even worse, when two producers are involved as shown in scenario 2 (Table 7.11). However, these side-effects caused by concurrency and the data structure used show that disconnection times might be beneficial since it reduces concurrent access attempts. The negative effects caused by mobility can be compensated by replication in the Producer/Consumer pattern as shown by Figure 7.13(a). There is no significant difference between the mobility cases and the stationary case. Figure 7.13(b) shows the negative effects of concurrency in detail. Better results can be achieved when using a different data structure instead of one single list, or by simply reducing the list size. Figure C.6 in the appendix depicts the advanced results when reducing the list size from 100 to 10 items.

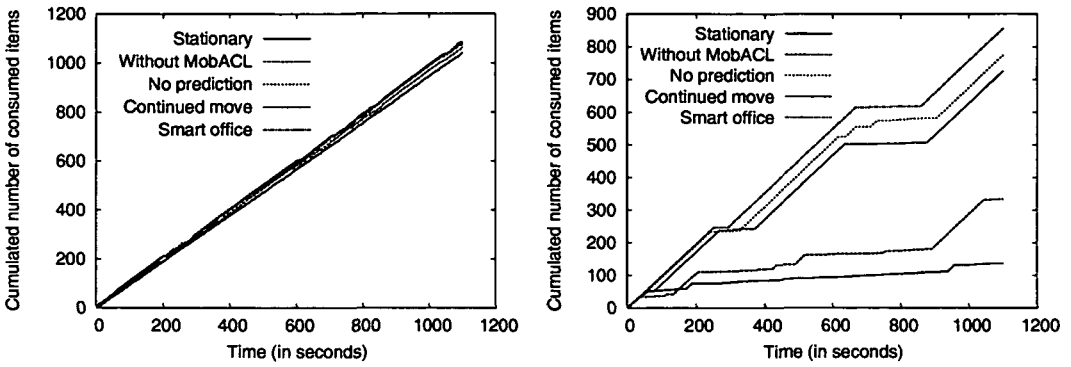


Figure 7.13: Referential coupling (Producer/Consumer): cumulated number of consumed items for (a) scenario 1 and (b) scenario 2

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	1041	1063	1077	1083	1075
Scenario 2	137	334	774	725	856

Table 7.11: Referential coupling: number of consumed items (Producer/Consumer)

Publisher/Subscriber

Two scenarios are used for the Publisher/Subscriber pattern in order to investigate the effects of mobility when a replicated process is added. *Scenario 1* is expanded by an additional publisher, while *scenario 2* is expanded by an additional subscriber.

Effectiveness. Scenario 1 shows that a mobile publisher can be compensated by replication effectively (Figure 7.14(a)). The subscriber receives notifications from the stationary publisher when the mobile publisher disconnects. In case both publishers update the value of interest, the subscriber does not receive significantly more notifications due to the limited read frequency of the subscriber (approximately 1 item per second). Since scenario 2 describes the situation of a mobile subscriber, here, in case of disconnections, the number of received updates per second decreases which is depicted as a slower increase of the cumulated function as depicted in Figure 7.14(b). The total number of updates received is summarized by Table 7.12.

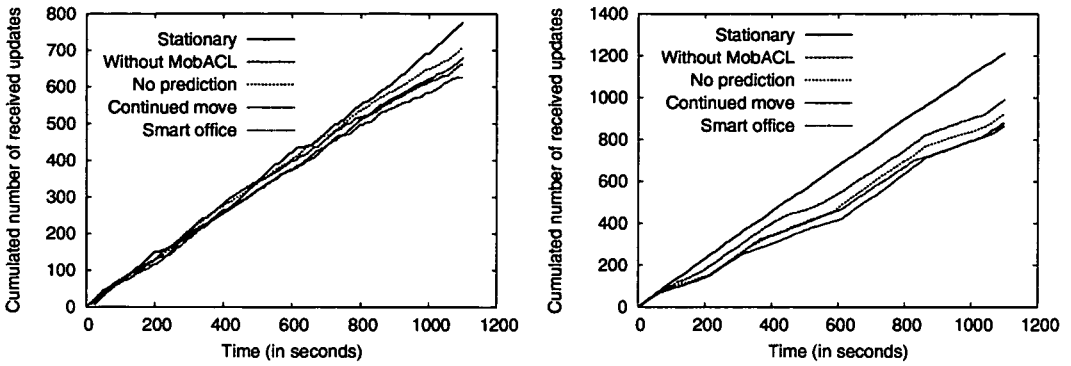


Figure 7.14: Referential coupling (Publisher/Subscriber): cumulated number of received updates for (a) scenario 1 and (b) scenario 2 (b)

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	777	664	708	680	626
Scenario 2	1211	990	926	882	882

Table 7.12: Referential coupling: number of received updates (Publisher/Subscriber)

Mailbox

For the referentially coupled but temporally uncoupled Mailbox pattern, three peers are used for evaluation. Two peers are executed on the stationary host and one peer is executed on the mobile host. Since the peers implement similar process types, only one scenario has to be investigated.

Effectiveness. Each of the three peer processes sends messages to one of the other remote peers and receive messages from the third peer. Here, concurrency phenomena have been observed when peers cannot get exclusive read access to their mailbox. These phenomena are shown in Figure 7.15 by slow increases of the cumulated number of received messages. Although the number of received messages is higher compared to the experiments carried out with only two peers, no peer is replaceable. This is due to the unique assignment between a mailbox and a particular peer (referentially coupled process). Table 7.13 summarizes the results.

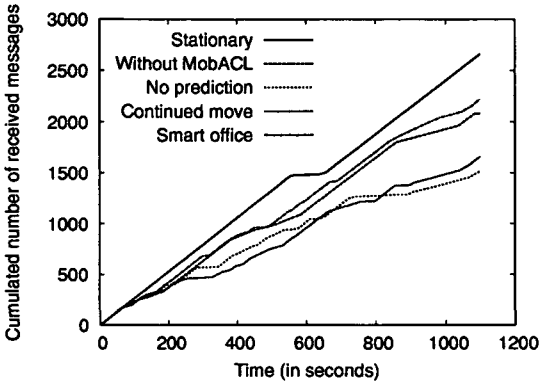


Figure 7.15: Referential coupling (Mailbox): cumulated number of received messages

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	2663	2082	1520	1660	2223

Table 7.13: Referential coupling: number of received messages (Mailbox)

Request/Answer

The Request/Answer pattern is both temporally and referentially coupled. Two scenarios are applied in order to study the effect of participant replication. *Scenario 1* consists of a mobile client which is replicated on the stationary host and a stationary server, while *scenario 2* consists of a mobile server which is replicated on the stationary host and ad stationary client. In this pattern, clients are addressed directly and cannot be replaced. In contrast, servers can be replaced.

Effectiveness. Scenario 1 shows that an added client benefits slightly if the mobile client disconnects (*no prediction*, *smart office*). Since a client may only generate up to one request per second and a server answers up to one request per second, the curves do not rise faster in case both clients are connected as shown by Figure 7.16(a). Furthermore, since answers are dedicated to specific clients, the mobile client cannot receive answers during disconnection times or when working on copies. For scenario 2, Figure 7.16(b) depicts that a replicated server can compensate the mobile server effectively. Table 7.14 summarizes the total number of received answers for each case.

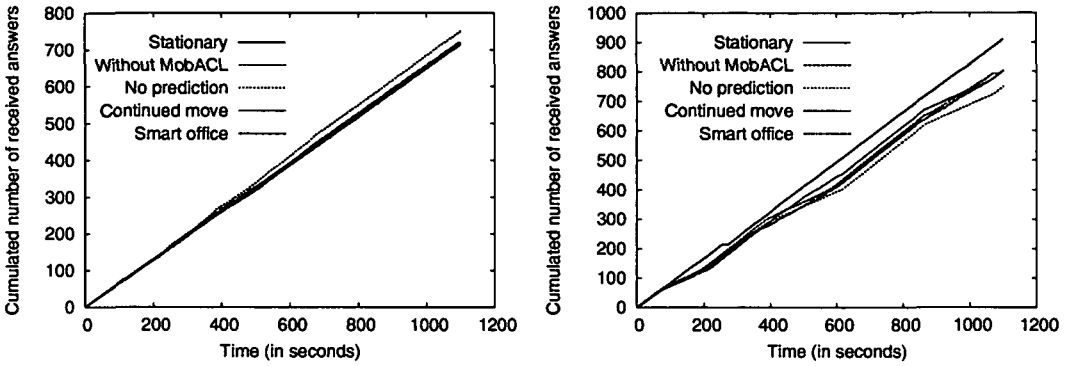


Figure 7.16: Referential coupling (Request/Answer): cumulated number of received answers for (a) scenario 1 and (b) scenario 2

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	720	714	750	717	722
Scenario 2	911	804	750	803	804

Table 7.14: Referential coupling: number of received answers (Request/Answer)

7.6 Discussion of the Experiments

In total, 70 simulation experiments have been carried out which have shown that the reference coordination patterns exhibit different potentials for making use of working on copies and replication of participants. These limiting factors are due to referential and temporal coupling characteristics of processes. By comparing the different mobility caused effects observed, beneficial approaches and limits can be argued.

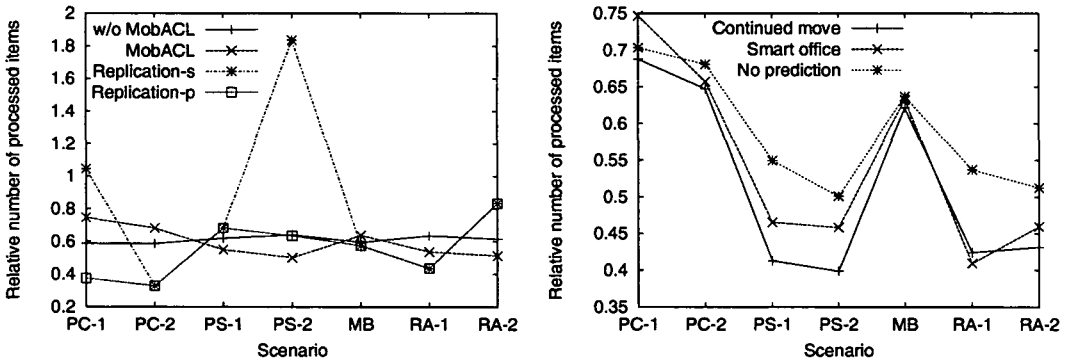
Effect of Mobility

Figure 7.17(a) depicts the effects of mobility for each scenario discussed in the previous sections by visualizing the relative number of processed items during the experiment (*w/o MobACL*), that is, the number of items processed while roaming related to the number of items processed in stationary mode.

The Request/Answer pattern based scenarios (scenario 1: *RA-1*, and scenario 2: *RA-2*) and the Publisher/Subscriber based scenarios (scenario 1: *PS-1*,

and scenario 2: *PS-2*) show slightly better results, in particular when the server or the publisher process is stationary. The Producer/Consumer pattern based scenarios (scenario 1: *PC-1*, and scenario 2: *PC-2*) and the Mailbox pattern based scenario (*Mb*) are affected similarly.

The differences between the scenarios based on the same coordination pattern are caused by different operation costs exhibited by the pattern participants. Due to the less powerful mobile host, effects are enforced for the participant executing more costly operations on the mobile host. Table 7.15 summarizes the results in row *w/o MobACL*.



(a) (b)

Figure 7.17: (a) Mobility caused effects and achievements, which are detailed by (b) comparing different MobACL cases

	PC-1	PC-2	PS-1	PS-2	MB	RA-1	RA-2
w/o MobACL	0.59	0.59	0.61	0.64	0.59	0.64	0.62
MobACL (best)	0.75	0.68	0.55	0.5	0.64	0.54	0.51
Replication-s	1.04	0.33	0.68	1.83	0.57	0.44	0.83
Replication-p	0.38	0.33	0.68	0.63	0.57	0.44	0.83
Continued move	0.69	0.65	0.41	0.4	0.62	0.42	0.43
Smart office	0.75	0.66	0.47	0.46	0.63	0.41	0.46
No prediction	0.7	0.68	0.55	0.5	0.64	0.54	0.51

Table 7.15: Comparison of mobility caused effects and achievements (relative number of processed items)

Compensation of Mobility Caused Failures

Compensation approaches of mobility caused failures have been carried out by using the MobACL approach and, additionally, by studying the potentials of replication. In the latter case, two different views are applicable. First, for anonymous coordination which applies for the Producer/Consumer and the Publisher/Subscriber case, a system view (*Replication-s*) considers the items processed by all consumers or, respectively, subscribers in the system.[7] Second, replication is viewed from the mobile process' perspective (*Replication-p*). Here, only the items successfully processed by the original process(es) are considered.

Table 7.15 lists the number of items processed while roaming in relation to the number of items consumed in the stationary mode for all scenarios described in the previous sections. For comparison reasons, a case without using MobACL (*w/o MobACL*), the best results achieved by MobACL - *MobACL (best)* - and both replication views are investigated. Except for the two replication views studied, all other cases do not use replicated processes.

Compared to the case without using MobACL, MobACL is able to improve throughput for the Producer/Consumer and the Mailbox pattern. In terms of improvement, *PC-1* shows the best results achievable by MobACL (Figure 7.17(a)). This result is due to proactive copying which allows the consumer to process items while a consumer working without MobACL cannot.

From a system point of view, replication of processes on the move can significantly improve throughput for the Producer/Consumer (*PC-1*) and the Publisher/Subscriber pattern (*PS-1, PS-2*), while it cannot compensate mobility related failures for referentially coupled processes, like the participants of the Mailbox pattern(*Mb*). In particular, a replicated consumer and a replicated subscriber can improve overall system throughput. The case of a replicated producer shows an anomaly due to concurrency conflicts which are mainly caused by the shared data structure selected (*PC-2*).

In case only items processed by the mobile process are considered (*Replication-p*), replication is significantly beneficial in the cases of a replicated publisher (*PS-1*) and a replicated server (*RA-2*). Although the Request/Answer coordination pattern is referentially coupled, the replication of the server process can partly compensate negative mobility caused effects.

Comparison of MobACL Cases

MobACL implements different approaches for state selection. One approach is based on a reactive model (*no prediction*), while the other two approaches use predictors in order to assure proactive state changes (*Continued move*, *Smart*

[7] The decision about the usefulness of the system view depends on the application semantics.

office). Figure 7.17(b) depicts the results achieved by the different cases for each scenario as detailed in the subsection addressing temporal coupling. Here again, the number of items processed is related to the number of items processed for the stationary case for each scenario.

In case of a mobile consumer and a stationary producer, the *Smart office* mobility pattern yields the best results (*PC-1*). For scenario 2 (*PC-2*), proactive copying is not needed because a producer does not need to copy existing items. Hence, the reactive approach performs best. The Mailbox pattern implements a combination of a producer and a consumer process type. Here, the benefits achieved by predictive mobility patterns for the consuming part are negated by the producing part.

The *Continued move* mobility pattern is not able to show better results than the reactive MobACL case due to the MobACL state machine. A safe algorithm is used which decides to work on copies whenever the wireless link state quality is below an SNR threshold value (link state *BAD*). Consequently, when continuously roaming from bad link quality to areas without any link connectivity, the process on the move is already working on copies and, thus, the reactive case does not come off badly. Table 7.15 summarizes the results achieved for each scenario.

Consequences of the Results Derived

The experiments have shown that mobility related wireless link quality degradation has significant effects on coordination patterns. Compared to the effects achieved in a fault-free setting (stationary case), all coordination patterns could only reach 59 – 64 percent of processed items. The difference between the coordination patterns is not significant.

The proposed mobility-aware layer (MobACL) is useful for temporally uncoupled patterns, where the effectiveness has been increased in the range of 5 – 16 percent. Best results have been achieved for the *Producer/Consumer* coordination pattern where the consumer process is roaming. For temporally coupled coordination patterns, MobACL deteriorates the effectiveness because of additional overhead added in the range of 6 – 14 percent. Consequently, MobACL has to be adapted in order to avoid unnecessary overhead for synchronous operations.

Referentially uncoupled patterns can be further supported by replicating processes. Since the coordination pattern participants compete and cooperate, replication is not always reasonable. In the cases where replication is semantically useful, cooperation can be supported and better results are achieved (except the anomaly for the *Producer/Consumer* where the roaming producer is replicated, caused by data structure related concurrency conflicts as described in Section 7.5.2). The achievements yield from 7 (*Publish/Subscriber* with replicated

publisher) to 21 percent (*Request/Answer* with a replicated server). In cases where replication is not reasonable but the pattern is temporally uncoupled, the use of MobACL is particularly suitable.

Finally, the evaluation of the mobility models used have shown, how a reliable predictor using a mobility pattern which describes the movement in an optimal way outperforms a weaker predictor. The results have further shown, that the MobACL, which is already very sensitive to bad network conditions, cannot benefit from a weak predictor (*Continued Move*). Cautious mechanisms, like working already on copies in case of bad link quality which can be exploited when the connection breaks, are sufficient in such cases.

Chapter 8

Conclusion

This thesis presented three contributions to the field of mobile computing based on the shared data space paradigm. First, in order to evaluate the effects of movement caused failures in significant coordination scenarios, the manifold of different interaction types has been reduced by means of coordination patterns.

Second, based on a fault-hypothesis, mechanism have been proposed to tolerate two major drawbacks of mobile computing, that are, (i) weak wireless network conditions and (ii) disconnection periods. Since these mechanisms have to be invoked before bad network conditions are observed, proactive operations based on prediction of future wireless link conditions have been introduced. A prototypical implementation demonstrated the feasibility of the approach.

Third, the effects of mobility and the benefits and limits of the mobility-aware approach have been evaluated experimentally. The majority of the experiments has been carried out by means of distributed simulation of a specific movement pattern.

Modeling of Coordination Patterns

Coordination patterns, that are, software design patterns for coordination purposes, provide a useful abstraction from specific coordination scenarios. The classification scheme based on coupling of coordinating processes in terms of time and reference is commonly used for inter-process communication.

The descriptive template proposed by Gamma et al. [Gam95] for modeling software design patterns has been enhanced in order to visualize, to describe, and to compare coordination patterns in terms of temporal and referential coupling more precisely.

Visualization has been proposed by means of UML activity diagrams. From these diagrams, sequences of data space-based primitives have been derived to

describe the interactions between coordination pattern participants. In order to compare the patterns in terms of referential and temporal coupling, measures have been defined which include the number of blocking and non-blocking operations and the number of processes which cooperate anonymously.

The coordination modeling approach has been applied to eight major coordination patterns identified during a thorough literature survey. It has been demonstrated, how data space-based interactions can be modeled explicitly. Based on the measures introduced, the eight reference patterns have been compared quantitatively and assigned to one class (out of four classes according to the taxonomy used). In particular complex coordination patterns are categorized as referentially and temporally coupled patterns. Here, the measures introduced allow to distinguish further between these coupled processes.

Mobility-Aware Coordination

The approach of mobility-aware coordination is based on the assignment of locations to wireless link conditions. Movement in a particular area can be perceived as roaming between different wireless link states. By analyzing the *Signal to Noise Ration (SNR)* of the wireless link, logical link states have been extracted. Based on the fault-hypothesis, the failure occurrence and compensation actions are detailed for each logical link state. Weak wireless link conditions and disconnection times have been identified as failure causes of interest.

In order to achieve accurate fault tolerance, the *Mobility-Aware Coordination Layer (MobACL)* has been introduced. This layer extracts the current logical link state and predicts future link state conditions as well as future retention periods which makes the approach *mobility-aware*. A prediction module has been introduced which allows to integrate different mobility predictors. For reference purpose, Markov models and a mobility model based on smart personalization, like a person's calendar entries (*Smart Office Mobility Pattern*), have been proposed. The Markov based predictor assumes that the moving person's velocity remains rather constant and that a person continues to walk in a given direction (*Continued Move Mobility Pattern*).

The MobACL has been designed to hide network conditions from the application by means of working on copies in case of weak wireless link quality. Mechanisms assuring accurate and proactive copying and release of data locks, and consistent synchronization upon reconnection have been proposed.

The modular and extensible reference implementation of the mobility-aware approach uses the shared object based middleware CORSO, wireless link observation by means of the ORINOCO WLAN client, and the Java programming language. Due to the taxonomy derived for coordination patterns, four representative coordination patterns, that are *Producer/Consumer*, *Publisher/Subscriber*,

Mailbox, and *Request/Answer*, have been implemented using the MobACL.

Evaluation Results

Experimental evaluation has been carried out in order to discuss mobility related coordination failures, the mobility-aware compensation operations proposed by different MobACL cases which are in particular suitable for temporally uncoupled processes, and compensation by means of process replication for referentially uncoupled processes. All experiments have been carried out under the same timing and movement schedule by means of a distributed simulator.

Evaluating Temporal Coupling

The first set of experiments has been carried out in order to observe the failures caused by mobility and to evaluate the benefits achievable by MobACL and the layer's limits. Compared to the stationary case, significant decrease of processed items has been observed during periods of weak link quality or disconnection. For all four coordination patterns, the effectiveness decreased to approximately 60 percent (compared to the effects achieved in stationary mode).

MobACL proved to compensate these negative effects partially for the temporally uncoupled coordination patterns. For example, in the *Producer/Consumer* case observing a mobile consumer and a stationary producer, 75 percent have been achieved, and for the *Mailbox* pattern, the rate could also be increased to 64 percent. For temporally coupled coordination patterns, that are, *Publisher/Subscriber* and *Request/Answer*, MobACL could not increase throughput. Unnecessary state changes caused impairments of throughput. Depending on the quality of the mobility predictor, the results achieved by proactive fault tolerance vary for temporally uncoupled processes. In case copying of data items is not important, reactive MobACL behavior has appeared to be sufficient.

Evaluating Referential Coupling

Referential coupling of processes has been addressed by a second set of experiments. Here, the process executing on the mobile device has been replicated on the stationary site. For the referentially coupled reference patterns, that are, *Mailbox* and *Request/Answer*), no achievements have been observed. However, by replicating the server of the referentially coupled *Request Answer* pattern, throughput of client requests have increased to 83 percent. Beside cooperative redundant processing, these experiments exhibit increased concurrency conflicts while interacting. For example, in the *Producer/Consumer* case with a replicated producer, the throughput achieved is worse than in the case without replication.

For the *Publish/Subscribe* pattern, the total number of processed items increased.

Future Work

In order to use and advance the results achieved, primarily two directions are most appealing and promising. First, the study and modeling of coordination patterns can be advanced in terms of variations and coordination failures. Second, ongoing optimization of the predictors by means of mobility modeling, tracing of movement, and extraction of mobility patterns can further improve the usefulness of the approach and the approach's applicability to application specific services.

In addition to the results achieved for investigating coordination scenarios by means of reference coordination patterns, it is reasonable to exploit these patterns in more detail. In order to improve a coordination pattern's usability, configuration facilities and fail-over concepts have to be included into the models. Hence, multiple variants of the patterns can be derived easily. Additionally, fault tolerance mechanisms reflecting the semantics of the pattern's coordination mechanisms can be specified, like synchronization actions in case of reconnections.

Mobility models have been used in this work as predictors for future link conditions and retention periods. Generating movement traces in real world scenarios in terms of place, retention, velocity, direction, and acceleration has been beyond the scope of this thesis. However, interpreting such traces by new mobility models which are capable of describing and learning about user movements in a scalable and fast way, is another important issue for future work.

List of Figures

List of Tables

Bibliography

[Abd04] W. Abdelsalam and Y. Ebrahim. *Managing Uncertainty: Modeling Users in Location-Tracking Applications. IEEE Pervasive Computing*, volume 3(3):pages 60–65, 2004.

[Add97] M.D. Addlesee, A.H. Jones, F. Livesey, and F.S. Samaria. *The ORL Active Floor. IEEE Personal Communications*, volume 4(5):pages 35–41, 1997.

[Age02] Agere. *Wireless LAN PC Card (Extended).* Product Sheet, Agere Systems Inc., 555 Union Boulevard, Room 30L-15P-BA, Allentown, PA 18109-3286, USA, 2002.

[All90] A.O. Allen. *Probability, Statistics, and Queueing Theory with Computer Science Applications.* Academic Press Professional, Inc., 1990.

[Ang02] B. Angerer. *Space Based Computing: J2EE bekommt Konkurrenz aus dem eigenen Lager. OBJEKT Spektrum*, volume 4:pages 47–49, 2002.

[Ang04] B. Angerer. *Mainstream Grid Computing: Softwareentwicklungen zur globalen Vernetzung. OBJEKT Spektrum*, volume 6:pages 38–42, 2004.

[Ash03] D. Ashbrook and T. Starner. *Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. Personal and Ubiquitous Computing*, volume 7(5):pages 275–286, 2003.

[Bak95] A.V. Bakre and B.R. Badrinath. *M-RPC: A Remote Procedure Call Service for Mobile Clients.* In *1st Annual International Conference on Mobile Computing and Networking*, pages 97–110. ACM Press, 1995.

[Bak96] A.V. Bakre and B.R. Badrinath. *Reworking the RPC Paradigm for Mobile Clients. Mobile Networking and Applications*, volume 1(4):pages 371–385, 1996.

[Bak00] A. Bakker, E. Amade, G. Ballintiijn, I. Kuz, P. Verkaik, I. van der Wijk, M. van Steen, and A.S. Tanenbaum. *The Globe Distribution Network*. In *USENIX Annual 2000 Technical Conference (FREENIX Track)*, pages 141–152. USENIX Association, 2000.

[Bel99] F. Bellifemine, A. Poggi, and G. Rimassa. *JADE - A FIPA-Compliant Agent Framework*. In *4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, pages 97–108. World Scientific, 1999.

[Bel00] P. Bellavista, A. Corradi, and C. Stefanelli. *A Mobile Agent Infrastructure for Terminal, User, and Resource Mobility*. In *IEEE/IFIP Network Operations and Management Symposium*, pages 877–890. IEEE Communications Society, 2000.

[Bel01] P. Bellavista and C. Stefanelli. *Mobile Agent Middleware for Mobile Computing*. *IEEE Computer*, volume 34(3):pages 73–81, 2001.

[Bel03a] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. *Policy-Driven Binding to Information Resources in Mobility-Enabled Scenarios*. In *4th International Conference on Mobile Data Management*, pages 212–229. Springer-Verlag, 2003.

[Bel03b] F. Bellotti, R. Berta, A. de Gloria, E. Ferretti, and M. Margarone. *Designing Mobile Games for a Challenging Experience of the Urban Heritage*. In *9th International Euro-Par Conference*, pages 1129–1136. Springer-Verlag, 2003.

[Bel04] P. Bellavista, A. Corradi, and C. Gianelli. *Mobility Prediction for Mobile Agent-Based Service Continuity in the Wireless Internet*. In *1st International Conference on Mobility Aware Technologies and Applications*, pages 1–12. Springer-Verlag, 2004.

[Ber01] F. Bergenti and A. Poggi. *LEAP: A FIPA Platform for Handheld and Mobile Devices*. In *Agent Theories, Architectures, and Languages Conference*, pages 436–446. Springer-Verlag, 2001.

[Bir84] A.D. Birrell and B.J. Nelson. *Implementing Remote Procedure Calls*. *ACM Transactions on Computer Systems*, volume 2(1):pages 39–59, 1984.

[Bis03] P. Bishop and N. Warren. *JavaSpaces in Practice*. Addison-Wesley, 2003.

[Bla01] G.S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski. *The Design and Implementation of Open ORB 2. IEEE Distributed Systems Online (online)*, volume 2(6), 2001.

[Bor02] J. Borchers, M. Ringel, J. Tyler, and A. Fox. *Stanford Interactive Workspaces: A Framework for Physical and Graphical User Interface Prototyping. IEEE Wireless Communications*, volume 9(6):pages 64–69, 2002.

[Bow03] K. Bowers, K. Mills, and S. Rose. *Self-Adaptive Leasing for Jini.* In *1st IEEE International Conference on Pervasive Computing and Communications*, page 539. IEEE Computer Society Press, 2003.

[Bru03a] M. Brunato and R. Battiti. *PILGRIM: A Location Broker and Mobility-Aware Recommendation System.* In *1st IEEE International Conference on Pervasive Computing and Communications*, page 265. IEEE Computer Society Press, 2003.

[Bru03b] D. Bruneo, M. Scarpa, A. Zaia, and A. Puliafito. *Communication Paradigms for Mobile Grid Users.* In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 669. IEEE Computer Society Press, 2003.

[Buk93] O. Bukhres, A. Elmagarmid, and e. Kühn. *Implementation of the Flex Transaction Model. IEEE Data Engineering Bulletin*, volume 16(2):pages 28–32, 1993.

[Bur04] I. Burcea, H.A. Jacobsen, E. de Lara, V. Methusamy, and M. Petrovic. *Disconnected Operation in Publish/Subscribe Middleware.* In *2004 IEEE International Conference on Mobile Data Management*, page 39. IEEE Computer Society Press, 2004.

[Bus01] N. Busi, P. Ciancarini, R. Gorrieri, and G. Zavattaro. *Coordination of Internet Agents*, chapter 1: Coordination Models: A Guided Tour, pages 6–24. Springer-Verlag, 2001.

[Bus02] N. Busi, A. Rowstron, and G. Zavattaro. *State- and Event-Based Reactive Programming in Shared Dataspaces.* In *5th International Conference on Coordination Models and Languages*, pages 111–124. Springer-Verlag, 2002.

[Cab98] C. Cabri, L. Leonardi, and F. Zambonelli. *Reactive Tuple Spaces for Mobile Agent Coordination.* In *2nd International Workshop on Mobile Agents*, pages 237–248. Springer-Verlag, 1998.

[Cab00a] G. Cabri, L. Leonardi, and F. Zambonelli. *MARS: A Programmable Coordination Architecture for Mobile Agents. IEEE Internet Computing*, volume 4(4):pages 26–35, 2000.

[Cab00b] G. Cabri, L. Leonardi, and F. Zambonelli. *Mobile Agent Coordination Models for Internet Applications. IEEE Computer*, volume 33(2):pages 82–89, 2000.

[Cab03] G. Cabri, L. Leonardi, M. Mamei, and F. Zambonelli. *Location-Dependent Services for Mobile Users. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, volume 33(6):pages 667–681, 2003.

[Cam97] A.T. Campbell. *Mobiware: QoS-Aware Middleware for Mobile Multimedia Communications*. In *IFIP TC6: 7th International Conference on High Performance Networking VII*, pages 166–183. Chapman & Hall, Ltd., 1997.

[Cam99] S. Campadello and K. Raatikainen. *Agents in Personal Mobility*. In *1st Int. Workshop on Mobile Agents for Telecommunication Applications*, pages 359–374. World Scientific, 1999.

[Cam00] S. Campadello, H. Helin, O. Koskimies, and K. Raatikainen. *Wireless Java RMI*. In *4th International Enterprise Distributed Object Computing Conference*, pages 25–28. IEEE Computer Society Press, 2000.

[Cam02] T. Camp, J. Boleng, and V. Davies. *A Survey of Mobility Models for Ad Hoc Network Research. Wireless Communications and Mobile Computing*, volume 2(5):pages 483–502, 2002.

[Cap02] L. Capra, G.S. Blair, C. Mascolo, W. Emmerich, and P. Grace. *Exploiting Reflection in Mobile Computing Middleware. ACM SIGMOBILE Mobile Computing and Communications Review*, volume 6(4):pages 34–44, 2002.

[Cap03] L. Capra, W. Emmerich, and C. Mascolo. *CARISMA: Context-Aware Reflective Middleware System for Mobile Applications. IEEE Transactions on Software Engineering*, volume 29(10):pages 929–945, 2003.

[Cay02] E. Cayirci, H. Akademileri, Y. Levent, and I. F. Akyildiz. *User Mobility Pattern Scheme for Location Update and Paging in Wireless Systems. IEEE Transactions on Mobile Computing*, volume 1(3):pages 236–247, 2002.

[Caz02] W. Cazzola. *mChaRM: Reflective Middleware with a Global View of Communications. IEEE Distributed Systems Online (online)*, volume 3(2), 2002.

[Cha99] D. Chalmers and M. Sloman. *A Survey of Quality of Service in Mobile Computing Environments. IEEE Communications Surveys (online)*, volume 2(2), 1999.

[Che03a] X. Chen, Y. Chen, and F. Rao. *An Efficient Spatial Publish/Subscribe System for Intelligent Location-Based Services.* In *2nd International Workshop on Distributed Event-Based Systems*, pages 1–6. ACM Press, 2003.

[Che03b] Y. Chen, K. Schwan, and D. Zhou. *Opportunistic Channels: Mobility-Aware Event Delivery. Lecture Notes in Computer Science*, volume 2672:pages 182–201, 2003.

[Che03c] C. Cheng, R. Jain, and E. van den Berg. *Wireless Internet Handbook*, chapter 11: Location Prediction Algorithms for Mobile Wireless Systems, pages 245–263. CRC Press, 2003.

[Cil03] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A.P. Buchmann. *Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware.* In *2nd International Workshop on Distributed Event-Based Systems*, pages 1–8. ACM Press, 2003.

[Con04] D. Conan, C. Taconet, D. Ayed, L. Chateigner, N. Kouici, and G. Bernard. *A Pro-Active Middleware Platform for Mobile Environments.* In *IASTED International Conference on Software Engineering*, page 701. Acta Press, 2004.

[Cou01] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems - Concepts and Design.* Pearson Education, 3rd edition, 2001.

[Cug01] G. Cugola, E. Di Nitto, and A. Fuggetta. *The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. IEEE Transactions on Software Engineering*, volume 27(9):pages 827–850, 2001.

[Dav98] N. Davies, A. Friday, S. Wade, and G. Blair. *L^2imbo: A Distributed Systems Platform for Mobile Computing. ACM Mobile Networks and Applications*, volume 3(2):pages 143–156, 1998.

[Dro97] R. Droms. *RFC 2131: Dynamic Host Configuration Protocol.* Request for Comment, Internet Society - IETF, 1775 Wiehle Ave., Suite 102, Reston, VA 20190-5108, USA, 1997.

[Duo04] H.H. Duong, A. Dadej, and S. Gordon. *Proactive Context Transfer in WLAN-Based Access Networks*. In *2nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, pages 61–70. ACM Press, 2004.

[Dür04] F. Dürr, N. Hönle, D. Nicklas, C. Becker, and K. Rothermel. *NEXUS – A Platform for Context-Aware Applications*. Informatikbericht 317, Fernuniversität Hagen, 1.GI/ITG KuVS Fachgespräch: Ortsbezogene Anwendungen und Dienste, Fernuniversität in Hagen, 58084 Hagen, Germany, 2004.

[Dus03] S. Dustar, H. Gall, and M. Hauswirth. *Software-Architekturen*. Springer-Verlag, 2003.

[dW03] N. Van den Wijngaert and C. Blondia. *A Location Augmented Low Latency Handoff Scheme for Mobile IP: OPNET Modeling and Simulation*. In *OPNETWORK 2003 (online)*. OPNET Technologies Inc., 2003.

[Edw99] K. Edwards. *Core JINI*. Prentice Hall, 1999.

[Eli99] F. Eliassen, A. Andersen, G. Blair, F. Costa, G. Coulson, V. Goebel, O. Hansen, T. Kristensen, T. Plagemann, H. O. Rafaelsen, K. B. Saikoski, and W. Yu. *Next Generation Middleware: Requirements, Architecture, and Prototypes*. In *7th IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 60–68. IEEE Computer Society Press, 1999.

[Fan04] S.H. Fan, T.Y. Xiao, and C. Guo. *A Model of the Expanded Grey Theory*. In *2004 IEEE International Conference on Networking, Sensing, and Control*, pages 339–342. IEEE Computer Society Press, 2004.

[Far04] U. Farooq, E.W. Parsons, and S. Majumdar. *Performance of Publish/Subscribe Middleware in Mobile Wireless Networks*. In *4th International Workshop on Software and Performance*, pages 278–289. ACM Press, 2004.

[Fer03] A. Ferscha. *Coordination in Pervasive Computing Environments*. In *12th IEEE International Workshops on Enabling Infrastructure for Collaborative Enterprises*, page 3. IEEE Computer Society Press, 2003.

[Fie03] L. Fiege, F.C. Gärtner, O. Kasten, and A. Zeidler. *Supporting Mobility in Content-Based Publish/Subscribe Middleware*. In *Middleware 2003*, pages 103–122. Springer-Verlag, 2003.

[Fis85] M.J. Fischer, N.A. Lynch, and M.S. Paterson. *Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM*, volume 32(2):pages 374–382, 1985.

[Fos96] I. Foster, C. Kesselman, and S. Tuecke. *The Nexus Approach to Integrating Multithreading and Communication. Journal of Parallel and Distributed Computing*, volume 37:pages 70–82, 1996.

[Fos97] I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications and High Performance Computing*, volume 11(2):pages 115–128, 1997.

[Fre97] B. Freisleben and T. Kielmann. *Coordination Patterns for Parallel Computing.* In *2nd International Conference on Coordination Languages and Models*, pages 414–417. Springer-Verlag, 1997.

[Fri02] D. Fritsch, D. Klinec, and S. Volz. *NEXUS: Integrating Data and Services for Mobile Users of Location Based Applications.* In *International Symposium on Networks for Mobility (CD ROM).* FOVUS, University of Stuttgart, 2002.

[Gam95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns.* Addison-Wesley, 1995.

[Gao04] X. Gao, G. Wu, and T. Miki. *End-to-end QoS Provisioning in Mobile Heterogeneous Networks. IEEE Wireless Communications*, volume 11(3):pages 24–34, 2004.

[GC02] F.J. Gonzalez-Castano, J. Vales-Alonso, M. Livny, E. Costa-Montenegro, and L. Anido-Rifon. *Condor: Grid Computing from Mobile Handheld Devices. ACM SIGMOBILE Mobile Computing and Communications Review*, volume 6(2):pages 18–27, 2002.

[Gei98] M. Geier, M. Steckermeier, U. Becker, F.J. Hauck, E. Meier, and U. Rastofer. *Support for Mobilty and Replication in the AspectIX Architecture.* In *12th European Conference on Object-Oriented Programming, Workshop Reader*, pages 325–326. Springer-Verlag, 1998.

[Gel85] D. Gelernter. *Generative Communication in Linda. ACM Transactions on Programming Languages and Systems*, volume 7(1):pages 80–112, 1985.

[Gel92] D. Gelernter and N. Carriero. *Coordination Languages and Their Significance. Communications of the ACM*, volume 35(2):pages 97–107, 1992.

[Gel02] H.W. Gellersen, A. Schmidt, and M. Beigl. *Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. Mobile Networks and Applications*, volume 7(5):pages 341–351, 2002.

[Gra04] S. Graumann and F. Neinert. *Monitoring Informationswirtschaft: 7. Faktenbericht - Infrastrukturelle Voraussetzungen.* Economy Research, TNS Infratest GmbH and Co. KG, Landsberger Strasse 338, 80687 - Munich, Germany, 2004.

[Gro00] G. Grossberger. *The Publish/Subscribe Coordination Design Pattern.* Master's thesis, Vienna University of Technology, 2000.

[Hat03] G. Hattori, C. Ono, S. Nishiyama, and H. Horiuchi. *Making Java-Enabled Mobile Phone As Ubiquitous Terminal by Lightweight FIPA Compliant Agent Platform.* In *1st IEEE International Conference on Pervasive Computing and Communication*, page 553. IEEE Computer Society Press, 2003.

[Hau01] F. J. Hauck, U. Becker, M. Geier, E. Meier, U. Rastofer, and M. Steckermeier. *AspectIX: A Quality-Aware Object-Based Middleware Architecture.* In *3rd IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 115–120. Kluwer, 2001.

[Haw03] K.A. Hawick and H.A. James. *Middleware for Context Sensitive Mobile Applications.* In *Australasian Information Security Workshop Conference on ACSW Frontiers - Volume 21*, pages 133–141. Australian Computer Society, Inc., 2003.

[Hay98] S. Hayden, C. Carrick, and Q. Yan. *Architectural Design Patterns for Multiagent Coordination.* In *3rd International Conference on Autonomous Agents.* 1998.

[Hea02] R. Headon and R. Curwen. *Movement Awareness for Ubiquitous Game Control. Personal and Ubiquitous Computing*, volume 6(5–6):pages 407–415, 2002.

[Hea03] R. Headon. *Movement Awareness for a Sentient Environment.* In *1st IEEE International Conference on Pervasive Computing and Communications*, page 99. IEEE Computer Society Press, 2003.

[Heu88] H. Heuser. *Lehrbuch der Analysis - Teil 2.* Teubner, 4th edition, 1988.

[Heu89] H. Heuser. *Lehrbuch der Analysis - Teil 1.* Teubner, 6th edition, 1989.

[Hit02] M. Hitz and G. Kappel. *UML @ Work*. dpunkt, 2nd edition, 2002.

[Hua04] Y. Huang and H. Garcia-Molina. *Publish/Subscribe in a Mobile Environment. Wireless Networks*, volume 10(6):pages 643–652, 2004.

[Hum03] K.A. Hummel, M. Ofner, B. Fiser, J.A. Mata Rodriguez, and M. Seewald. *Context-Aware M-Learning Based on Meta-Learning Units*. In *2003 International Conference on Computer Aided Learning (CD ROM)*. Kassel University Press, 2003.

[Hum04a] K.A. Hummel. *Mobility-Aware Adaptation of Coordination Patterns. Network and Information Systems*, volume 9(2):pages 87–106, 2004.

[Hum04b] K.A. Hummel, R. El-Berry, B. Klein, and H. Hlavacs. *Tracking Physical and Digital Learning Interactions in Ubiquitous Computing Environments*. In *2004 International Conference on Computer Aided Learning (CD ROM)*. Kassel University Press, 2004.

[Hum04c] K.A. Hummel and H. Hlavacs. *Industrial Information Technology*, chapter 23: Mobile IP Routing, pages 23–1 – 23–9. CRC Press, 2004.

[Hwa99] K. Hwang and Z. Xu. *Scalable Parallel Computing*. McGraw-Hill, 1999.

[Hwa04] J. Hwang and P. Aravamudham. *Middleware Services for P2P Computing in Wireless Grid Networks. IEEE Internet Computing*, volume 8(4):pages 40–46, 2004.

[IBM03] IBM. *WebSphere MQ Intercommunication*. International Business Machines Corporation, 4th edition, 2003.

[IEE99] IEEE. *ANSI/IEEE Standard 802.11*. Specification, IEEE Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA, 1999.

[IEE02] IEEE. *ANSI/IEEE Standard 802.15*. Specification, IEEE Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA, 2002.

[Imi94] T. Imielinski and B.R. Badrinath. *Mobile Wireless Computing: Challenges in Data Management. Communications of the ACM*, volume 37(10):pages 18–28, 1994.

[Jär04] R. Järvensivu, R. Pitkänen, and T. Mikkonen. *Object-Oriented Middleware for Location-Aware Systems*. In *2004 ACM Symposium on Applied Computing*, pages 1184–1190. ACM Press, 2004.

[Joh03] B. Johanson and A. Fox. *Extending Tuplespaces for Coordination in Interactive Workspaces. The Journal of Systems and Software*, volume 69(3):pages 243–266, 2003.

[Jos95] A.D. Joseph, A.F. de Lespinasse, J.A. Tauber, D.K. Gifford, and M.F. Kashoek. *Rover: A Toolkit for Mobile Information Access.* In *15th ACM Symposium on Operating Systems Principles*, pages 156–171. ACM Press, 1995.

[Jul04] C. Julien, J. Payton, and G.-C. Roman. *Reasoning About Context-Awareness in the Presence of Mobility. Electronic Notes in Theoretical Computer Science*, volume 97:pages 259–276, 2004.

[Kin97] G. Kinczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. *Aspect-Oriented Programming.* In *11th European Conference on Object Oriented Programming*, page 220. Springer-Verlag, 1997.

[Kin01] T. Kindberg and J. Barton. *A Web-Based Nomadic Computing System. Computer Networks: The International Journal of Computer and Telecommunications Networking*, volume 35(4):pages 443–456, 2001.

[Kob00] H. Kobayashi, S.Z. Yu, and B.L. Mark. *An Integrated Mobility and Traffic Model for Resource Allocation in Wireless Networks.* In *3rd ACM International Workshop on Wireless Mobile Multimedia*, pages 39–47. ACM Press, 2000.

[Kon00] F. Kon, M. Román, P. Liu, J. Mao, T. Ymane, L. C. Magalhaes, and R. H. Campell. *Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB.* In *IFIP/ACM International Conference on Distributed Systems Platforms*, pages 121–143. Springer-Verlag, 2000.

[Kop97] H. Kopetz. *Real-Time Systems.* Kluwer, 1997.

[Kou04] I.Z. Koukoutsidis and M.E. Theologou. *A Combination of Optimal Partitioning and Location Prediction to Assist Paging in Mobile Cellular Networks. International Journal of Wireless Information Networks*, volume 11(3):pages 123–129, 2004.

[Küh98a] E. Kühn. *Introduction: How to Approach the Virtual Shared Memory Paradigm. Journal of Parallel and Distributed Computing Practices (available online)*, volume 1(3), 1998.

[Küh98b] E. Kühn and G. Nozicka. *Post-Client/Server Coordination Tools. Coordination Technology for Collaborative Applications*, volume 1364:page 231, 1998.

[Küh01] E. Kühn. *Coordination system.* Patent EP0929864 B1, European Patent Office, D–80298 Munich, 2001.

[Küh02] E. Kühn. *Coordinated Shared Object Spaces v1.2.* White Paper, TECCO Software Entwicklung, Prinz Eugen Strasse 58, A1040 Vienna, 2002.

[Kum03] M. Kumar, B.A. Shirazi, S.K. Das, B.Y. Sung, D. Levine, and M. Singhal. *PICO: A Middleware Framework for Pervasive Computing. IEEE Pervasive Computing*, volume 2(3):pages 72–79, 2003.

[Kur03] S. Kurkovsky and Bhagyavati. *Modeling a Computational Grid of Mobile Devices as a Multi-Agent System.* In *2003 International Conference on Artificial Intelligence*, pages 36–44. CSREA Press, 2003.

[Kur04] S. Kurkovsky, Bhagyavati, and A. Ray. *Modeling a Grid-Based Problem Solving Environment for Mobile Devices.* In *International Conference on Information Technology: Coding and Computing*, page 135. IEEE Computer Society Press, 2004.

[Lam82] L. Lamport, R. Shostak, and M. Pease. *The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems*, volume 4(3):pages 382–401, 1982.

[Lan98] D.B. Lange. *Mobile Objects and Mobile Agents: The Future of Distributed Computing?.* In *12th European Conference on Object-Oriented Programming*, page 1. Springer-Verlag, 1998.

[Lap92] J.C. Laprie, editor. *Dependability: Basic Concepts and Terminology.* Springer-Verlag, 1992.

[Las97] G.v. Laszewski, S. Fitzgerald, I. Foster, C. Kesselman, W. Smith, and S. Tuecke. *A Directory Service for Configuring High-Performance Distributed Computations.* In *6th IEEE Symposium on High-Performance Distributed Computing*, pages 5–8. IEEE Computer Society Press, 1997.

[Led99] T. Ledoux. *OpenCorba: A Reflective Open Broker.* In *2nd International Conference on Meta-Level Architectures and Reflection*, page 197. Springer-Verlag, 1999.

[Leh99] T.J. Lehman, S.W. McLaughry, and P. Wycko. *T Spaces: The Next Wave.* In *32 Annual Hawaii International Conference on System Sciences*, page 8037. IEEE Computer Society Press, 1999.

[Lei95] R.D. Leitch. *Reliability Analysis for Engineers.* Oxford University Press, 1995.

[Lew96] M. Lewis and A. Grimshaw. *The Core Legion Object Model.* In *High Performance Distributed Computing*, page 551. IEEE Computer Society Press, 1996.

[Lil97] M. Liljeberg, K. Raatikainen, M. Evans, S. Furnell, N. Maumon, E. Veldkamp, B. Wind, and S. Trigila. *Using CORBA to Support Terminal Mobility.* In *Global Convergence of Telecommunications and Distributed Object Computing*, pages 59–67. IEEE Computer Society Press, 1997.

[Lin03] C. Lindemann and O.P. Waldhorst. *Exploiting Epidemic Data Dissemination for Consistent Lookup Operations in Mobile Applications. Mobile Computing and Communications Review*, volume 8(2):pages 44–56, 2003.

[Luc02] Lucent. *Orinoco PC Card.* User Manual, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974-0636, USA, 2002.

[Mam03] M. Mamei, F. Zambonelli, and L. Leonardi. *Programming Coordinated Motion Patterns with the TOTA Middleware.* In *9th International Euro-Par Conference*, pages 1027–1037. Springer-Verlag, 2003.

[Mar97] J.G. Markoulidakis, G.L. Lyberopoulos, D.F. Tsirkas, and E.D. Sykas. *Mobility Modeling in Third-Generation Mobile Telecommunications Systems. IEEE Personal Communications*, volume 4(4):pages 41–56, 1997.

[Mas02a] C. Mascolo, L. Capra, and W. Emmerich. *Middleware for Mobile Computing (A Survey).* In *Lecture Notes in Computer Science*, volume 2497, pages 20–58. Springer-Verlag, 2002.

[Mas02b] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. *XMIDDLE: A Data-Sharing Middleware for Mobile Computing. Wireless Personal Communications*, volume 21(1):pages 77–103, 2002.

[Mas04] C. Mascolo, L. Capra, and W. Emmerich. *Middleware for Communications*, chapter 11: Principles of Mobile Computing Middleware, pages 261–280. John Wiley & Sons, 2004.

[May03] R. Mayrhofer, H. Radi, and A. Ferscha. *Recognizing and Predicting Context by Learning from User Behavior*. In *International Conference on Advances in Mobile Multimedia*, pages 25–35. Austrian Computer Society, 2003.

[McK04] L.W. McKnight, J. Howison, and S. Bradner. *Wireless Grids: Distributed Resource Sharing by Mobile, Nomadic, and Fixed Devices. IEEE Internet Computing*, volume 8(4):pages 24–31, 2004.

[Men04] D.A. Menasce. *QoS-Aware Software Components. IEEE Internet Computing*, volume 8(2):pages 91–93, 2004.

[Mic00] Microsoft. *Understanding Universal Plug and Play*. White paper, Microsoft Corporation, One Microsoft Way, WA 98052-6399, Redmond, USA, 2000.

[Moo03] J.M. Moon, M.Y. Yun, Y.J. Kim, and S.H. Kim. *History-Based Adaptive QoS Provisioning in Mobile IP Networks*. In *2003 Global Telecommunications Conference*, pages 3483–3487. IEEE Computer Society Press, 2003.

[Müh04] G. Mühl, A. Ulbrich, K. Hermann, and T. Weis. *Disseminating Information to Mobile Clients Using Publish-Subscribe. IEEE Internet Computing*, volume 8(3):pages 46–53, 2004.

[Mur01] A.L. Murphy, G.P. Picco, and G.-C. Roman. *LIME: A Middleware for Physical and Logical Mobility*. In *21st International Conference on Distributed Computing Systems*, page 0524. IEEE Computer Society Press, 2001.

[Mus04a] M. Musolesi. *Designing a Context-Aware Middleware for Asynchronous Communication in Mobile Ad Hoc Environments*. In *1st International Doctoral Symposium on Middleware*, pages 304–308. ACM Press, 2004.

[Mus04b] M. Musolesi, C. Mascolo, and S. Hailes. *Adapting Asynchronous Messaging Middleware to Ad Hoc Networking*. In *2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pages 121–126. ACM Press, 2004.

[Nah01] L.K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li. *QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. IEEE Communications Magazine*, volume 39(11):pages 140–148, 2001.

[Nat02] A. Natrajan, A. Nguyen-Tuong, M. Humphrey, M. Herrick, B.P. Clarke, and A. Grimshaw. *The Legion Grid Portal. Concurrency*

and Computation: Practice and Experience, volume 14(13-15):pages 1365–1394, 2002.

[Nic04] D. Nicklas and B. Mitschang. *On Building Location Aware Applications Using an Open Platform Based on the NEXUS Augmented World Model. Software and Systems Modeling*, volume 3(4):pages 303–313, 2004.

[Nob97] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. *Agile Application-Aware Adaptation for Mobility.* In *16th ACM Symposium on Operating System Principles*, pages 276–287. ACM Press, 1997.

[OMG04a] OMG. *Common Object Request Broker Architecture (CORBA/IIOP) - v3.0.3*. Specification, Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, USA, 2004.

[OMG04b] OMG. *Wireless Access and Terminal Mobility in CORBA*. Specification, Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, USA, 2004.

[Pas99] B. Pascoe. *Salutation Architectures and the Newly Defined Service Discovery Protocols from Microsoft and Sun*. White paper, Salutation Consortium, http://www.salutation.org/ (online), 1999.

[Pea80] M. Pease, R. Shostak, and L. Lamport. *Reaching Agreement in the Presence of Faults. Journal of the ACM*, volume 27(2):pages 228–234, 1980.

[Ped04] A. Peddemors, H. Zandbelt, and M. Bargh. *A Mechanism for Host Mobility Management Supporting Application Awareness.* In *2nd International Conference on Mobile Systems, Applications, and Services*, pages 231–244. ACM Press, 2004.

[Per96] C.E. Perkins. *RFC 2002: IP Mobility Support*. Request for Comment, Internet Society - IETF, 1775 Wiehle Ave., Suite 102, Reston, VA 20190-5108, USA, 1996.

[Per98] C.E. Perkins. *Mobile IP Design Principles*. Prentice Hall, 1998.

[Pic97] G.P. Picco. *Mobile Agents: An Introduction. Microprocessors and Microsystems*, volume 25(2):pages 65–74, 1997.

[Pic98] G.P. Picco. *μCODE: A Lightweight and Flexible Mobile Code Toolkit.* In *2nd International Workshop on Mobile Agents*, pages 160–171. Springer-Verlag, 1998.

[Pic00] G.P. Picco, A.L. Murphy, and G.-C. Roman. *Developing Mobile Computing Applications with LIME*. In *22nd International Conference on Software Engineering*, pages 766–769. ACM Press, 2000.

[Pop03] A. Popovici, A. Frei, and G. Alonso. *A Proactive Middleware Platform for Mobile Computing*. In *Middleware 2003*, pages 455–473. Springer-Verlag, 2003.

[Pos80] J. Postel. *RFC 768: User Datagram Protocol*. Request for Comment, Internet Society - IETF, 1775 Wiehle Ave., Suite 102, Reston, VA 20190-5108, USA, 1980.

[Pos81] J. Postel. *RFC 793: Transmission Control Protocol*. Request for Comment, Internet Society - IETF, 1775 Wiehle Ave., Suite 102, Reston, VA 20190-5108, USA, 1981.

[Rom99] M. Romberg. *The UNICORE Architecture: Seamless Access to Distributed Resources*. In *8th IEEE International Symposium on High Performance Computing*, page 44. IEEE Computer Society Press, 1999.

[Rom00a] G.-C. Roman, G.P. Picco, and A.L. Murphy. *Software Engineering for Mobility: A Roadmap*. In *Conference on the Future of Software Engineering*, pages 241–258. ACM Press, 2000.

[Rom00b] M. Román and R.H. Campbell. *Gaia: Enabling Active Spaces*. In *9th ACM SIGOPS European Workshop: Beyond the PC*, pages 229–234. ACM Press, 2000.

[Rom01] M. Román, R.H. Campbell, and F. Kon. *Reflective Middleware: From Your Desk to Your Hand. IEEE Distributed Systems Online (online)*, volume 2(5), 2001.

[Rom02] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. *A Middleware Infrastructure for Active Spaces. IEEE Pervasive Computing*, volume 1(4):pages 74–83, 2002.

[Rom03] M. Román and R.H. Campbell. *A Middleware-Based Application Framework for Active Space Applications. Lecture Notes in Computer Science*, volume 2672:pages 433–454, 2003.

[Ros01] E. Rosen, A. Viswanathan, and R. Callon. *RFC 3031: Multiprotocol Label Switching Architecture*. Request for Comment, Internet Society - IETF, 1775 Wiehle Ave., Suite 102, Reston, VA 20190-5108, USA, 2001.

[Roy04] A. Roy, S.K. Das, and A. Misra. *Exploiting Information Theory for Adaptive Mobility and Resource Management in Future Cellular Networks. IEEE Wireless Communications*, volume 11(4):pages 59–65, 2004.

[Sac04] V. Sacramento, M. Endler, H.K. Rubinsztejn, L.S. Lima, K. Goncalves, F.N. Nascimento, and G.A. Bueno. *MoCA: A Middleware for Developing Collaborative Applications for Mobile Users. IEEE Distributed Systems Online (online)*, volume 5(10), 2004.

[Sal04] A. Salovaara and A. Oulasvirta. *Six Modes of Proactive Resource Management: A User-Centric Typology for Proactive Behaviors.* In *3rd Nordic Conference on Human-Computer Interaction*, pages 57–60. ACM Press, 2004.

[Sam03] N. Samaan and A. Karmouch. *An Evidence-Based Mobility Prediction Agent Architecture.* In *5th International Workshop on Mobile Agents for Telecommunication Applications*, pages 230–239. Springer-Verlag, 2003.

[Sam04] F.A. Samimi, P.K. McKinley, S.M. Sadjadi, and P. Ge. *Kernel-Middleware Interaction to Support Adaptation in Pervasive Computing Environments.* In *2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pages 140–145. ACM Press, 2004.

[Sch95] A. Schill and S. Kümmel. *Design and Implementation of a Support Platform for Distributed Mobile Computing. Distributed Systems Engineering*, volume 2(3):pages 128–141, 1995.

[Sch04] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture – Volume 2.* John Wiley & Sons, 2004.

[Sho01] P.G. Shotsberger and R. Vetter. *Teaching and Learning in the Wireless Classroom. IEEE Computer*, volume 34(3):pages 110–111, 2001.

[Smi02] C.U. Smith and L.G. Williams. *Performance Solutions – A Practical Guide to Creating Responsive, Scalable Software.* Addison-Wesley, 2002.

[Smi03] O. Smirnova, P. Eerola, T. Ekelöf, M. Ellert, J.R. Hansen, A. Konstantinov, B. Konya, J.L. Nielsen, F. Ould-Saada, and A. Wäänänen. *The NorduGrid Architecture and Middleware for Scientific Applications. Lecture Notes in Computer Science*, volume 2657:pages 264–273, 2003.

[Son03] M.B. Song, J.H. Ryu, and C.-S. Hwang. *A Mobility-Aware Location Update Protocol to Track Users in Location-Based Services. Lecture Notes in Computer Science*, volume 2762:pages 352–359, 2003.

[Sor04] C.-F. Sorensen, M. Wu, T. Sivaharan, G.S. Blair, P. Okanda, A. Friday, and H. Duran-Limon. *A Context-Aware Middleware for Applications in Mobile Ad Hoc Environments.* In *2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pages 107–110. ACM Press, 2004.

[Ste03] I. Stepanov, J. Hähner, C. Becker, J. Tian, and K. Rothermel. *A Meta-Model and Framework for User Mobility in Mobile Networks.* In *11th IEEE International Conference on Networks*, pages 231–238. IEEE Computer Society Press, 2003.

[Sun99] Sun. *Jini Architecture Overview.* White paper, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, USA, 1999.

[Sun00] Sun. *JavaSpaces Service Specification – Version 1.1.* Specification, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, USA, 2000.

[Sun02] Sun. *Java Messaging Service – Version 1.1.* Specification, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, USA, 2002.

[Sun03] Sun. *Java 2 Platform Enterprise Edition (J2EE) Specification – Version 1.4.* Specification, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, USA, 2003.

[Tak03] K. Takasugi, M. Nakamura, S. Tanaka, and M. Kubota. *Seamless Service Platform for Following a User's Movement in a Dynamic Network Environment.* In *1st IEEE International Conference on Pervasive Computing and Communications*, page 71. IEEE Computer Society Press, 2003.

[Tan02] A.S. Tanenbaum and M. van Steen. *Distributed Systems - Principles and Paradigms.* Prentice Hall, 2002.

[Tan03] A.S. Tanenbaum. *Computer Networks.* Prentice Hall, 4th edition, 2003.

[Tar03] S. Tarkoma, J. Kangasharju, and K. Raatikainen. *Client Mobility in Rendezvous-Notify.* In *2nd International Workshop on Distributed Event-Based Systems*, pages 1–8. ACM Press, 2003.

[TEC04] TECCO. *CORSO v3.3*. Tutorial, TECCO Software Entwicklung, Prinz Eugen Strasse 58, A1040 Vienna, 2004.

[Ter95] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. *Managing Update Conflicts in Bayou, a Weakly Connected Storage System*. In *15th ACM Symposium on Operating Systems Principles*, pages 172–183. ACM Press, 1995.

[Viv03] S. Vivek, K. Tso, and D. de Roure. *Mobile Link Services with MQSeries Everyplace*. In *1st IEEE International Conference on Pervasive Computing and Communications*, page 359. IEEE Computer Society Press, 2003.

[vL01] K. van Laerhoven, K.A. Aidoo, and S. Lowette. *Real-Time Analysis of Data from Many Sensors with Neural Networks*. In *5th IEEE International Symposium on Wearable Computers*, page 115. IEEE Computer Society Press, 2001.

[vS99] M. van Steen, P. Homburg, and A.S. Tanenbaum. *Globe: A Wide-Area Distributed System. IEEE Concurrency*, pages 70–78, 1999.

[Whi97] J.E. White. *Software Agents*, chapter 19: Mobile Agents, pages 437–472. MIT Press, 1997.

[Wyc98] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. *T Spaces. IBM Systems Journal*, volume 37(3):pages 454–474, 1998.

[Zac02a] S. Zachariadis, L. Capra, C. Mascolo, and W. Emmerich. *XMIDDLE: Information Sharing Middleware for a Mobile Environment*. In *24th International Conference on Software Engineering*, page 712. ACM Press, 2002.

[Zac02b] S. Zachariadis, C. Mascolo, and W. Emmerich. *Exploiting Logical Mobility in Mobile Computing Middleware*. In *22nd International Conference on Distributed Computing Systems Workshops*, page 385. IEEE Computer Society Press, 2002.

[Zai04] Z.R. Zaidi and B.L. Mark. *Mobility Estimation for Wireless Networks Based on an Autoregressive Model*. In *2004 Global Telecommunications Conference*, pages 3405–3409. IEEE Computer Society Press, 2004.

Appendix A

Analytical Discussion of Coupling Degrees

In Section 4.4, eight coordination patterns are discussed and compared by discrete functions describing temporal ($D_{TempUncoupled}$) and referential coupling ($D_{RefUncoupled}$) as given by Table 4.4. In order to derive extreme values and thus, the range of these functions, they are transformed into continuous functions which are differentiable on the definition set. The functions are defined as follows:

$$f(x) : D \rightarrow [0,1], \text{ where } D \subseteq \mathbb{R}^n, \ n \geq 1. \tag{A.1}$$

The minimum and maximum values are calculated for each pattern. In the non-trivial cases, it is first shown that no local extreme value exists by analyzing the first-order partial derivative, which exists for every function. It is shown that these derivatives are always non-zero within the definition set, which is a necessary condition for extreme values [Heu89]. Second, the limits of the functions are calculated for the boundary points, curves, or planes.

Since the functions for the *Producer/Consumer* and the *Mailbox* pattern are constant functions, no additional discussion is needed.

Publisher/Subscriber

According to Table 4.4, the degree of temporal coupling is given as follows on $[1,\infty) \rightarrow [0,1]$:

$$D_{TempUncoupled\_PS}(k) = \frac{1+2k}{1+3k}. \tag{A.2}$$

Since this function decreases monotonically, by applying the rule of de

l'Hospital [Heu89], p. 287, the minimum is calculated by

$$Minimum\ D_{TempUncoupled\_PS} = \lim_{k\to\infty} \frac{1+2k}{1+3k} = 2/3. \tag{A.3}$$

The maximum evaluates as follows ($k = 1$):

$$Maximum\ D_{TempUncoupled\_PS} = 3/4. \tag{A.4}$$

$D_{RefUncoupled\_PS}()$ is a constant function. The minimum and maximum values evaluate to 1 (see Table 4.4).

Master/Worker

$D_{TempUncoupled\_MW}()$ is a constant function. The maximum and minimum values evaluate to 3/4 (see Table 4.4).

According to Table 4.4, the function describing referential coupling is defined on $[1,\infty) \times [1,\infty) \rightarrow [0,1]$ as follows:

$$D_{RefUncoupled\_MW}(c_1, c_2) = \frac{c_2}{c_1 + c_2}. \tag{A.5}$$

The first-order partial derivatives are denoted as:

$$D_{RefUncoupled\_MW\_c_1}(c_1, c_2) = -\frac{c_2}{(c_1 + c_2)^2}, \text{ and} \tag{A.6}$$

$$D_{RefUncoupled\_MW\_c_2}(c_1, c_2) = \frac{1}{c_1 + c_2} - \frac{c_2}{(c_1 + c_2)^2} = \frac{c_1}{(c_1 + c_2)^2}. \tag{A.7}$$

Both derivatives never equate to zero on the definition set. Hence, no local extreme value exists. In order to derive border curve extreme values, the function has to be studied on the border of the definition area. In case of $D_{RefUncoupled\_MW}()$, the minimum and maximum extreme value equal the minimum 0 and the maximum 1 of the value set, respectively. The minimum evaluates as follows ($c_2 = 1$):

$$Minimum\ D_{RefUncoupled\_MW} = \lim_{c_1\to\infty} \frac{1}{c_1 + 1} = 0. \tag{A.8}$$

By applying the rule of de l'Hospital, the maximum evaluates as follows ($c_1 = 1$):

$$Maximum\ D_{RefUncoupled\_MW} = \lim_{c_2\to\infty} \frac{c_2}{1 + c_2} = 1. \tag{A.9}$$

Request/Answer

$D_{TempUncoupled\_RA}()$ is a constant function. The minimum and maximum values evaluate to 1/2 (see Table 4.4).

According to Table 4.4, the function describing referential coupling is defined on $[1, \infty) \times [1, \infty) \rightarrow [0, 1]$ as follows:

$$D_{RefUncoupled\_RA}(c_1, c_2) = \frac{c_2}{c_1 + c_2}. \tag{A.10}$$

The function and definition set is the same as for the *Master/Worker* coordination pattern. Hence, the minimum and maximum values are given as:

$$Minimum\ D_{RefUncoupled\_RA} = 0, \quad \text{and} \tag{A.11}$$

$$Maximum\ D_{RefUncoupled\_RA} = 1. \tag{A.12}$$

Proxy

According to Table 4.4, the function describing temporal coupling is defined on $[1, \infty) \times [1, \infty) \rightarrow [0, 1]$ as follows:

$$D_{TempUncoupled\_Pr}(c_1, c_2) = \frac{4c_1 + 2c_2}{3c_1 + 2c_2}. \tag{A.13}$$

The first-order partial derivatives are calculated as follows:

$$\begin{aligned} D_{TempUncoupled\_Pr\_c_1}(c_1, c_2) &= \frac{4}{3c_1 + 2c_2} - \frac{3(4c_1 + 2c_2)}{(3c_1 + 2c_2)^2} = \\ &= \frac{2c_2}{(3c_1 + 2c_2)^2}, \quad \text{and} \end{aligned} \tag{A.14}$$

$$\begin{aligned} D_{TempUncoupled\_Pr\_c_2}(c_1, c_2) &= \frac{2}{3c_1 + 2c_2} - \frac{2(4c_1 + 2c_2)}{(3c_1 + 2c_2)^2} = \\ &= -\frac{2c_1}{(3c_1 + 2c_2)^2}, \end{aligned} \tag{A.15}$$

which never equate to zero on the definition set. Hence, no local extreme value exists. In order to derive border curve extreme values, the function has to be studied on the border of the definition area.

Since the function is monotonic in both dimensions, only the border points of the area have to be investigated. These points are $(1, 1)$, $(1, c_2 \rightarrow \infty)$, $(c_1 \rightarrow$

$\infty, 1)$, and $(c_1 \to \infty, c_2 \to \infty)$. The results of this investigation are used to derive the minimum as follows $((c_1 \to \infty, 1))$:

$$Minimum\ D_{TempUncoupled\_Pr} = \lim_{c_1 \to \infty} \frac{3c_1 + 2}{4c_1 + 2} = 3/4. \tag{A.16}$$

The maximum evaluates as follows $((1, c_2 \to \infty))$:

$$Maximum\ D_{TempUncoupled\_Pr} = \lim_{c_2 \to \infty} \frac{3 + 2c_2}{4 + 2c_2} = 1. \tag{A.17}$$

The other two points are no candidates for neither the maximum nor the minimum. In case of border point $(1, 1)$, the function evaluates to $5/6$, which can be neither the minimum, nor the maximum. The double limit of the function does not exist, otherwise the iterated limits would be the same despite of their order (see [Heu89], p. 568):

$$\lim_{c_2 \to \infty} \lim_{c_1 \to \infty} \frac{3c_1 + 2c_2}{4c_1 + 2c_2} = 3/4 \quad \neq \quad \lim_{c_1 \to \infty} \lim_{c_2 \to \infty} \frac{3c_1 + 2c_2}{4c_1 + 2c_2} = 1. \tag{A.18}$$

Since the double limit does not exist, it cannot be an extreme value.

$D_{RefUncoupled\_Pr}()$ is a constant function. The minimum and maximum values evaluate to 0 (according to Table 4.4).

Consensus

According to Table 4.4, the function describing temporal coupling is defined on $[2, \infty) \to [0, 1]$ as follows:

$$D_{TempUncoupled\_Cs}(c_1) = \frac{2}{1 + c_1}. \tag{A.19}$$

This function is monotonically decreasing. Hence, by analyzing the border points of the definition set, the minimum value is calculated as follows $(c_1 \to \infty)$:

$$Minimum\ D_{TempUncoupled\_Cs} = \lim_{c_1 \to \infty} \frac{2}{1 + c_1} = 0, \quad \text{and} \tag{A.20}$$

the maximum is calculated accordingly $(c_1 = 2)$:

$$Maximum\ D_{TempUncoupled\_Cs} = 2/3. \tag{A.21}$$

$D_{RefUncoupled\_Cs}()$ is a constant function. The minimum and maximum values evaluate to 0 (according to Table 4.4).

Broker

According to Table 4.4, the function describing temporal coupling is defined on $[1,\infty) \times [1,\infty) \times [1,\infty) \rightarrow [0,1]$ as follows:[1]

$$D_{TempUncoupled\_Br}(c_1, c_3, k) = \frac{4c_1k + c_1c_3k + c_3}{8c_1k + c_1c_3k + c_3}. \tag{A.22}$$

The first-order partial derivatives on the definition area are denoted as:

$$\begin{aligned} D_{TempUncoupled\_Br\_c_1}(c_1, c_3, k) = &\frac{4k + c_3k}{8c_1k + c_1c_3k + c_3} + \\ &+ \frac{(4c_1k + c_1c_3k + c_3)(-1)(8k + c_3k)}{(8c_1k + c_1c_3k + c_3)^2} = \\ &= -\frac{4c_3k}{(8c_1k + c_1c_3k + c_3)^2}, \text{ and} \end{aligned} \tag{A.23}$$

$$\begin{aligned} D_{TempUncoupled\_Br\_c_3}(c_1, c_3, k) = &\frac{c_1k + 1}{8c_1k + c_1c_3k + c_3} + \\ &+ \frac{(4c_1k + c_1c_3k + c_3)(-1)(c_1k + 1)}{(8c_1k + c_1c_3k + c_3)^2} = \\ &= \frac{(c_1k + 1)4c_1k}{(8c_1k + c_1c_3k + c_3)^2}. \end{aligned} \tag{A.24}$$

Similarly to Equation A.23, it follows:

$$D_{TempUncoupled\_Br\_k}(c_1, c_3, k) = -\frac{4c_1c_3}{(8c_1k + c_1c_3k + c_3)^2}. \tag{A.25}$$

Since all derivatives never equate to zero on the definition set, no local extreme value exists. In order to derive global extreme values, the function has to be studied on the border of the definition area. Since the function is monotonic in all dimensions, only the border points of the definition set have to be investigated as expressed by the following case distinction:

$$D_{TempUncoupled\_Pr}(c_1, c_3, k) = \tag{A.26}$$

$$= \begin{cases} c_1, c_3, k = 1: & 3/5 \\ c_1, c_3 = 1, k \rightarrow \infty: & \lim_{k\rightarrow\infty} \frac{5k+1}{9k+1} = 5/9 \\ c_1, k = 1, c_3 \rightarrow \infty: & \lim_{c_3\rightarrow\infty} \frac{4+2c_3}{8+2c_3} = 1 \\ c_1 =\rightarrow \infty, c_3, k = 1: & \lim_{c_1\rightarrow\infty} \frac{5c_1+1}{9c_1+1} = 5/9 \\ c_1 = 1, c_3, k =\rightarrow \infty: & \lim_{(c_3\rightarrow\infty, k\rightarrow\infty)} \frac{4k+c_3k+c_3}{8k+c_3k+c_3}, \text{ (n.e.)} \\ c_1, k =\rightarrow \infty, c_3 = 1: & \lim_{(c_1\rightarrow\infty, k\rightarrow\infty)} \frac{4c_1k+c_1k+1}{8c_1k+c_1k+1} = 5/9, \text{ if exists} \\ c_1, c_3 =\rightarrow \infty, k = 1: & \lim_{(c_1\rightarrow\infty, c_3\rightarrow\infty)} \frac{4c_1+c_1c_3+c_3}{8c_1+c_1c_3+c_3}, \text{ (n.e.)} \\ c_1, c_3, k =\rightarrow \infty: & \lim_{(c_1\rightarrow\infty, c_3\rightarrow\infty, k\rightarrow\infty)} \frac{4c_1k+c_1c_3k+c_3}{8c_1k+c_1c_3k+c_3}, \text{ (n.e.)} \end{cases}$$

[1] The variable names are chosen consistently to Table 4.4.

As argued for the *Proxy* coordination pattern, the multiple limit (or, in our case, the triple limit) cannot exist, if the iterated limits are not equal *(n.e.)*. In case the iterated limits are equal, the multiple limit equals this value, if it exists.[2] Consequently, the minimum and maximum values are given as follows:

$$Minimum\ D_{TempUncoupled\_Br} = 5/9, \text{ and} \tag{A.27}$$

$$Maximum\ D_{TempUncoupled\_Br} = 1. \tag{A.28}$$

The degree of referential coupling is given according to Table 4.4. The function is defined on $[1,\infty) \times [1,\infty) \times [1,\infty) \rightarrow [0,1]$ as follows:

$$D_{RefUncoupled\_Br}(c_1, c_2, c_3) = \frac{c_2 + c_3}{c_1 + c_2 + c_3}. \tag{A.29}$$

The first-order partial derivatives on the definition area are denoted as:

$$D_{RefUncoupled\_Br\_c_1}(c_1, c_2, c_3) = -\frac{c_2 + c_3}{(c_1 + c_2 + c_3)^2}, \text{ and} \tag{A.30}$$

$$\begin{aligned} D_{RefUncoupled\_Br\_c_2}(c_1, c_2, c_3) &= D_{RefUncoupled\_Br\_c_3}(c_1, c_2, c_3) = \\ &= \frac{1}{c_1 + c_2 + c_3} - \frac{c_2 + c_3}{(c_1 + c_2 + c_3)^2} = \\ &= \frac{c_1}{(c_1 + c_2 + c_3)^2}. \end{aligned} \tag{A.31}$$

Since all derivatives never equate to zero on the definition set, no local extreme value exists. In order to derive global extreme values, the function has to be studied on the border of the definition set. In case of $D_{RefUncoupled\_Br}()$, the minimum and maximum values equal the maximum 1 and the minimum 0 of the value set, respectively. The minimum value evaluates as follows ($c_2 = 1$, $c_3 = 1$):

$$Minimum\ D_{RefUncoupled\_Br} = \lim_{c_1 \rightarrow \infty} \frac{2}{c_1 + 2} = 0. \tag{A.32}$$

The maximum evaluates as follows ($c_1 = 1$, $c_2 = 1$):

$$Maximum\ D_{RefUncoupled\_Br} = \lim_{c_3 \rightarrow \infty} \frac{1 + c_3}{2 + c_3} = 1. \tag{A.33}$$

[2]Note, that the existence and equality of the iterated limit does not imply the existence of a multiple limit.

Appendix B

Validating the Simulation

In order to model the wireless LAN link states for simulation purpose, three modes are identified. First, for excellent network conditions, the simulator does not alter the maximum data transfer rate achievable (that is, 11 Mbit/s). Second, for disconnected states, no operations can complete successfully. Finally, in order to simulate weak link quality, an appropriate operation success rate has to be used.

In order to derive this rate for weak link simulation, a reference coordination pattern is selected, that is, the *Producer/Consumer* pattern. The pattern's effectiveness while roaming physically is compared to the simulated case iteratively until the simulation shows satisfying results.

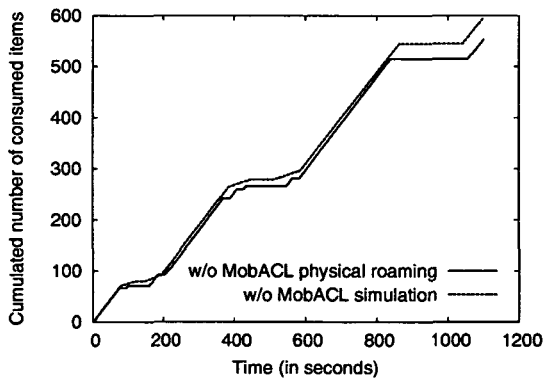


Figure B.1: Approximation of the physical roaming results by simulation using an operation success rate of 0.25 (Producer/Consumer)

Figure B.1 depicts the approximation of the physical roaming results by the simulation using an operation success rate of 0.25. This value has been chosen for all experiments. Although the physical roaming case differs from the simulation

slightly because of earlier connection loss at $t = 850$ seconds caused by imprecise movement, the curves show the same tendencies. Consequently, this specific configuration of the simulation is chosen as a reasonable approximation of the physical roaming behavior.

Appendix C

Additional Results for Experimental Evaluation

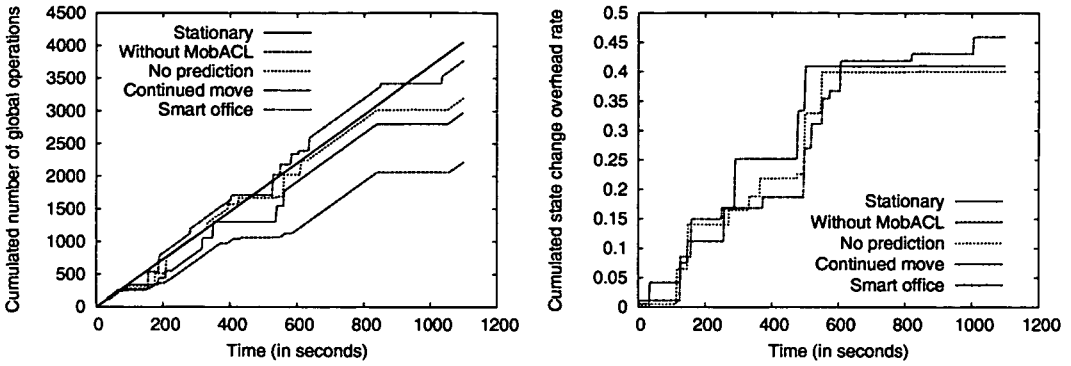
In addition to the results presented in Chapter 7, here additional evaluations are described briefly. For a definition of the measures and scenarios used, refer to Chapter 7.

C.1 Physical Roaming

Figure C.1(a) depicts the cumulated number of global space operations executed during the physical roaming experiments using the *Producer/Consumer* coordination pattern. Table C.1 details the total numbers of global space operations. Additionally, Figure C.1(b) depicts the cumulated number of overhead rates and Table C.1 details the overhead operations.

	Stationary	**Without MobACL**	**No Prediction**	**Continued Move**	**Smart Office**
Global op.	4044	2216	3198	2983	3767
Overhead rate	0	0	0.40	0.41	0.46

Table C.1: Physical roaming: number of global operations and state change overhead rates (Producer/Consumer)



(a) (b)

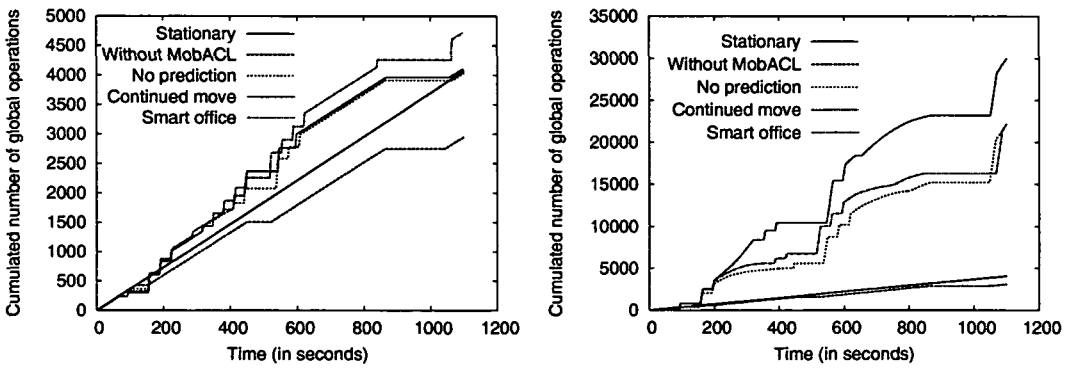
Figure C.1: Physical roaming (Producer/Consumer): (a) cumulated number of global operations and (b) cumulated state change overhead rates with one mobile consumer and one stationary producer

C.2 Simulation – Temporal Coupling

In this section, the global operations executed during the experiments used to evaluate temporal coupling are depicted.

Producer/Consumer

Figure C.2(a) and Figure C.2(b) depict the cumulated numbers of global space operations executed. Table C.2 details the total numbers of global space operations.



(a) (b)

Figure C.2: Temporal coupling (Producer/Consumer): cumulated number of global operations for (a) scenario 1 and (b) scenario 2

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	4074	2952	4037	4121	4731
Scenario 2	4074	2952	22171	30015	22198

Table C.2: Temporal coupling: number of global operations (Producer/Consumer)

Publisher/Subscriber

Figure C.3(a) and Figure C.3(b) depict the cumulated numbers of global space operations executed which are detailed by Table C.3.

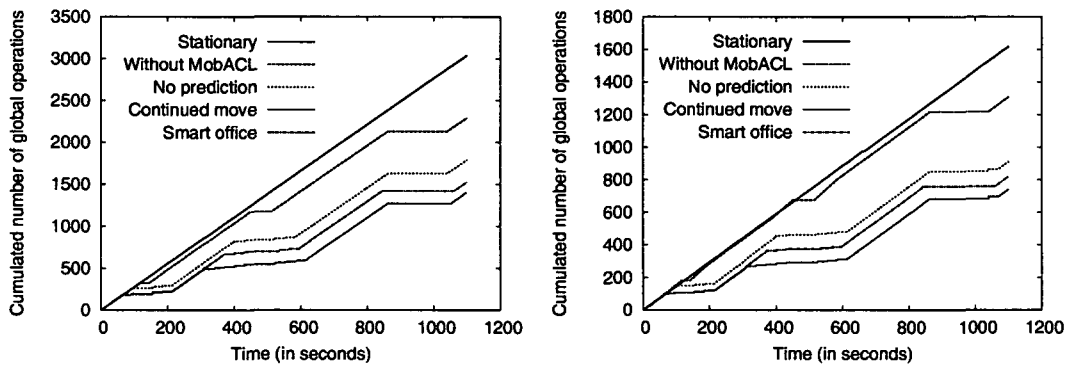


Figure C.3: Temporal coupling (Publisher/Subscriber): cumulated number of global operations for (a) scenario 1 and (b) scenario 2

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	3004	2293	1792	1408	1528
Scenario 2	1618	1309	911	741	1618

Table C.3: Temporal coupling: number of global operations (Publisher/Subscriber)

Mailbox

Figure C.4 depicts the cumulated numbers of global space operations executed. Table C.4 details the total numbers of global space operations.

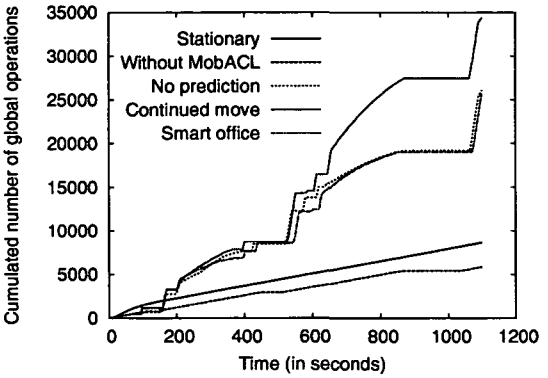


Figure C.4: Temporal coupling (Mailbox): cumulated number of global operations

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	8657	5857	26013	34397	25676

Table C.4: Temporal coupling: number of global operations (Mailbox)

Request/Answer

Figure C.5(a) and Figure C.5(b) depict the cumulated numbers of global space operations executed which is detailed in Table C.5.

Scenario	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 1	2704	2259	1582	1194	1139
Scenario 2	4805	3577	2688	2218	2342

Table C.5: Temporal coupling: number of global operations (Request/Answer)

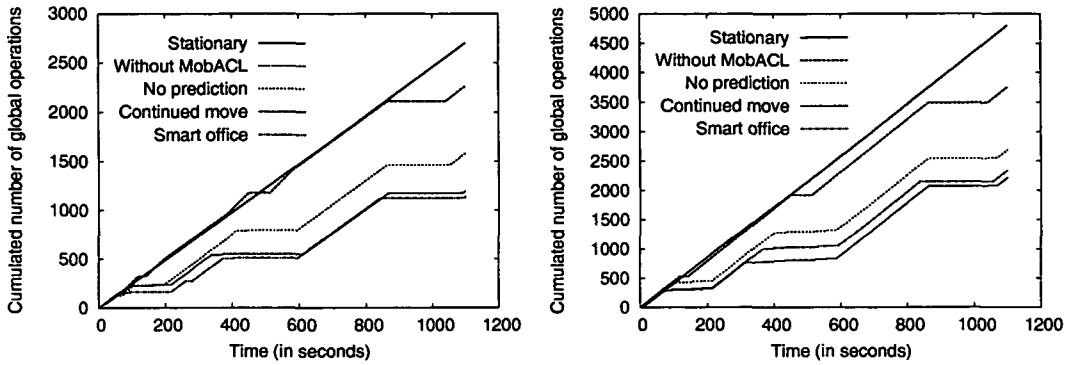


Figure C.5: Temporal coupling (Request/Answer): cumulated number of global operations for (a) scenario 1 and (b) scenario 2

C.3 Simulation – Referential Coupling

Producer/Consumer – Alteration of the Data Structure Used

The experiments carried out for a replicated producer have shown that the list data structure deteriorates the results achievable by applying replication due to increased concurrency conflicts. In order to demonstrate a possible reduction of conflicts by reducing the list size,[1] the list size is decreased to 10 for scenario 2. Figure C.6 depicts the effects and Table C.6 details the results.

	Stationary	Without MobACL	No Prediction	Continued Move	Smart Office
Scenario 2	942	818	1057	946	1049

Table C.6: Producer/Consumer with reduced list size (10): number of items consumed

[1]The original upper bound of the list size is 100.

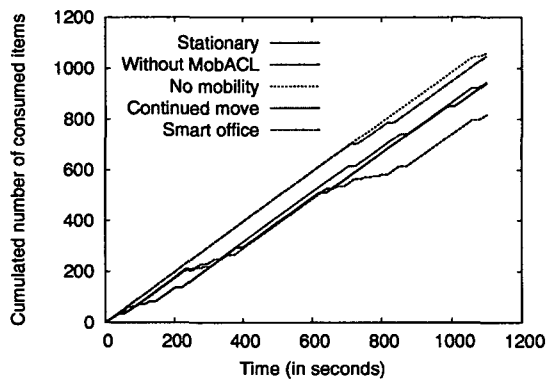


Figure C.6: Referential coupling (Producer/Consumer): cumulated number of items consumed using a list size of 10 - scenario 2

Curriculum Vitae

Karin Anna Hummel

2. Dezember 1970	geboren in Wien
1977 – 1981	Besuch der Volksschule Berzeliusgasse, 1210 Wien
1981 – 1989	Besuch des Gymnasiums, BG/BRG XXI Ödenburgerstrasse, 1210 Wien
Juni 1989	Matura mit ausgezeichnetem Erfolg
1989 – 1996	Studium der Informatik an der Technischen Universität Wien
November 1996	Sponsion zur Diplom-Ingenieurin der Informatik
1993 – 1998	Software Entwicklerin, Projektleiterin und Trainerin bei PSE/Siemens Österreich AG
seit 1998	Doktoratsstudium der technischen Wissenschaften an der Technischen Universität Wien
1998 – 2002	Universitätsassistentin am Institut für Distributed and Multimedia Systems der Universität Wien
seit 2002	Wissenschaftliche Mitarbeiterin am Institut für Distributed and Multimedia Systems der Universität Wien