A Service-Based Agent System Supporting Mobile Computing

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften
genehmigte Dissertation

vorgelegt von

Diplom-Informatiker Anthony Sang-Bum Park
aus Seoul (Süd-Korea)

Berichter:
Universitätsprofessor Dr. rer. nat. Otto Spaniol
Universitätsprofessorin Dr. rer. nat. Claudia Linnhoff-Popien

Tag der mündlichen Prüfung: 10. März 2004

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

An dieser Stelle möchte ich mich bei allen Personen bedanken, die durch Diskussionen, gemeinsame Projektarbeiten und Vorträge zum Gelingen dieser Arbeit beigetragen haben. Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Angestellter am Lehrstuhl für Informatik 4 der RWTH Aachen und hat mit Umwegen einen Abschluss gefunden.

Mein herzlicher und besonderer Dank gilt Prof. Dr. Spaniol für die kontinuierliche Betreuung und für den uneingeschränkten Freiraum bezüglich meiner Forschungsaktivitäten und den Forschungsprojekten. Ihm habe ich die wertvolle Zeit am Lehrstuhl und die Erfahrungen auf internationalen Forschungsprojekten und -veranstaltungen zu verdanken, wodurch unter anderem die Chance entstand, das eigene Unternehmen zu gründen.

Prof. Dr. Claudia Linnhoff-Popien danke ich für das Koreferat, aber vor allen Dingen für die förderlichen Gespräche und als wichtige Projektpartnerin während der Zeit in München.

Für die fachkundigen Gespräche mit den Kollegen und die nützlichen Arbeiten meiner Diplomanden und Studenten möchte ich mich hiermit ausdrücklich bedanken. Neben den fachbezogenen Ereignissen liegen mir die unzähligen freundschaftlichen Aktivitäten am Herzen, insbesondere für meinen unvergesslichen Junggesellenabend danke ich Axel und Steffen. Für die unzähligen Gespräche und Abende gilt der Dank des weiteren Jens, Roland, Helen, Kai, Ulrich, Peter, Carsten, Michael, Dirk, Frank, Christian, Rainer, Petra und Christiane.

Auch nach der Aachener Zeit in München haben mir Freunde wertvolle Beihilfe geleistet, Axel danke ich für das gemeinsame Forschungsprojekt, Steffen für die Mithilfe im Unternehmen und Marc für die gesamte Zusammenarbeit.

Nicht zuletzt gilt der Dank meinen Eltern für ihre besondere Unterstützung und meiner Frau Dong-Mi, die mir stets ein starker, mentaler Beistand ist und immer zum Gelingen der Dissertation beigetragen hat.

München, im März 2004

Anthony Sang-Bum Park

Contents

Chapter 1

Motivation and Outline

The exhaustive emergence of agent-based systems emphasizes that mobile agent technology discloses one of the most important and interesting computing paradigms since the object oriented design and client/server-based distributed systems. The information technology is enriched by the methodology that mobile agents bring along with them. The most general definition of an agent in computer science is probably "*any piece of continuously running software that communicates and cooperates with others*". Historically, two types of agents have been studied in different research communities: intelligent agents have been investigated quite extensively since 1951 in the *Artificial Intelligence* (AI) community, whereas mobile agents are more recent (taking the first relevant commercial product as reference, since 1994), mainly concerning distributed computing and communication.

The AI research has had many different goals: intelligent agents have been employed for problem-solving, (uncertain) knowledge and reasoning, acting logically (planning), and learning; for instance, robots learn about their environment or act on inconsistent information. They are considered to be cooperative or competitive, communicating, perceiving, and acting; accordingly to their protocols, they have been aimed towards coordination or negotiation. Simple intelligent software agents have already made their way into products as users' assistance learning their preferences, recognizing their handwriting, and filtering information, or as stationary agents backing routers, or as *softbots* designed to comb the world wide web - to name only a few. Mobile agents do not claim to be "intelligent", and in fact most mobile agents do not satisfy the definition of intelligence given by the AI community. In distributed systems, they are considered as an alternative to client/server technology, which enhance distributed applications by continuously running processes with code mobility. In other words, a mobile agent consists of a fragment of code and an execution state and it is able to migrate autonomously in a network of computers. Nonetheless, a transfer of intelligence into the network is apparently carried out with the agent technology.

The advantages mobile agents offer are the unique opportunities for structuring and implementing open distributed systems. The network infrastructure for mobile agents offers a high degree of flexibility and efficiency to the user by performing much of the work somewhere in the network, on the one hand, and enabling convenient application usage, on the other. The execution of agent-based applications, commonly for temporary use, takes place without having the awkward steps of creating source directories, configuration, and installation. Any agent-enabled computer can plug into the network and participate, the software that is needed comes from the net. Consequently, the most interesting property of mobile agents is referred to as *remote programming*. This approach differs fundamentally from classical client/server computing. For example, agent technology can help reduce required network resources by executing mobile agents close to resources they use instead of a remote access. Furthermore, the agents do not need any additional user intervention or involvement. Ideally, mobile agents act autonomously and asynchronously, and allow a convenient way of distributing lightweight applications and services, respectively.

A wide range of applications has been proposed for mobile agent technology, including for instance electronic commerce, telecommunication services, work flow, and network management. In particular, mobile agents promise an increasing number of new value added services for mobile computing. An important aspect not to be neglected is the explosive development of wireless and cellular networks, the growing popularity of the Internet, and the extreme success in miniaturization of mobile devices, which leads to a widespread wireless access to services. It already begins to emerge as the *World Wide Web* (WWW) becomes a *service network* and not only an information database. The ability of autonomous and disconnected operations of mobile agents makes them a high-potential technology for the 3rd generation mobile communication networks such as the *Universal Mobile Telecommunication System* (UMTS). Mobile agents may fulfil their task in the fixed network and return to the mobile device when the network quality allows a migration. However, along with the potential benefits, autonomy and disconnected operations of mobile agents in a wireless environment raise fundamental problems, which so far have found little attention in existing mobile agent systems. Autonomous work of agents in an open agent infrastructure requires mechanisms to enable search for other agents and services. Furthermore, inter-operable services for mobile agents in an open environment require more than a simple look up function, which must particularly be transparent to the user. The tracking of mobile agents in general and the handling of mobile agents, which are unable to reach their destination and their support in case of failures are subject to research.

The major objective of this thesis is to introduce a service-based mobile agent environment that demonstrates solutions for the support of autonomous, disconnected operations of mobile agents. Attention is paid extensively to the management of mobile agents at the boundary of wireless access networks. Based on the introduced architecture and the measurements performed in a local area network, another focus is on the analytical study of an envisaged network management scenario, which underlines the beneficial application of agent technology.

Outline of the Thesis

The foundation of this thesis is in particular the design and implementation of the agent system architecture and the infrastructure of the service-based agent network. This so called *Java Agent Environment* (JAE) has been developed and implemented since early 1996 during my research work at the *Rheinisch-Westfälische Technische Hochschule* (RWTH) Aachen. JAE is a framework for developing and deploying mobile agents and agent-based services. It contains the core technology for mobility, services, security, management, and thus, allows implementation of user defined agents. Distinguishing features of JAE are the distributed trading facility, the involvement of standardized directory services, and the strong support of mobile agents in wireless access networks.

In chapter two, the main components of mobile agent technology are introduced in detail and the components of a mobile agent system are outlined in general. The paradigm and the concepts, which come with this technology are explained. The definition of the agent terminology helps to understand the different recommendations and proposals of standardisation bodies such as *Foundation for Intelligent Physical Agents* (FIPA) and *Mobile Agent System Interoperability Facility* (MASIF). Chapter two concludes with a comparison and a survey of existing agent systems.

Chapter three gives an overview of the JAE architecture, its framework for developing and deploying mobile agents, and points out its novelties and improvements to the existing agent systems. Subsequently, the core technology for agent mobility, agent security, agent services, and agent communication is described. Special attention is paid to the service trading facilities introduced by the so-called *service center.* Implementation details, measurements, and programming examples to mediate the service-based agent programming paradigm will show the simplicity of creating mobile agents. A comprehensive description of mobile agent migration completes this chapter.

Chapter four begins with a brief introduction into cellular and wireless local area networks, which are the basis for the second research topic - the support of services in mobile computing. On the basis of the *Global System for Mobile Communication* (GSM), problems with mobile agents and requirements of the agent network and the agent system at the border of wireless access networks are described. An obvious solution to management of disconnected and malfunctioning mobile agents is given by the so-called *kindergarten* and *maintenance* concept, which are both realized with the service-based agent programming paradigm of JAE. In the second part of this chapter, user mobility and the support of personalized service access with mobile agents are discussed.

In chapter five, new strategies for network management approaches based on agents are discussed and their suitability for meeting current and future network management requirements is assessed. Mobile agents seem to be a solution for problems occurring in typical heterogene-

ous network organizations with their large and growing architecture, including a variety of LANs, WLANs, WANs, and distributed computing services and devices. The main part of this chapter describes the analytical models of the different management strategies regarding both packet loss probabilities of the network and failure probability of the network devices. It becomes more and more important to consider these aspects of failure due to the fact that wireless links and dynamically changing network topology are becoming increasingly dominant in today's and future networks.

Chapter six concludes the thesis with a summary and an outlook on further possible applications based on mobile agent technology.

Chapter 2

Mobile Agent System

The mobile agent paradigm has gained momentum within the last 10 years and the popularity of mobile agents and the interest in this technology has lead to developments and implementations of a variety of proprietary execution environments for mobile agents, so-called *mobile agent system*. On the one hand, these agent systems have a quite similar range of applications and are even written in the same programming language. On the other hand, the concepts and the scope are completely different although the agent systems are written in the same programming language and thus impede the communication and co-operation between agents in an assorted agent system network. Due to the fact that agents are executed on different physical hosts and operating systems, the forerunners of agent systems have been developed on interpreted languages such as *LISt Processing language* (Lisp, for references consult the Association of Lisp Users [LISP]), *Tool Command Language* (TCL, [TCL03, Ous98]), or *Python* [RoDr03]. Although, well-known agent system implementations have been developed in other programming languages like C/C++ such as the first commercial product from General Magic, *Telescript* [Whi94], the dominant language today for realizing the mobile agent technology is Java [Java03]. Unfortunately, the standardization efforts by non-profit organisations composed of companies and research institutes do not have a resounding success. At present, there are about 50 commercial as well as research agent systems listed (without claiming completeness), but in a heterogeneous system, none of the mobile agents could collaborate with another. Consequently, standardization is an ongoing and important process.

In this chapter mobile agent technology is introduced in detail and the components of a mobile agent system are outlined in general. The property of the paradigm and the concepts which come with this technology, are explained. Having defined the terminology that will be used with the *Java Agent Environment*, JAE, the recommendations and proposals of standardisation bodies are compared. A survey of existing agent systems establishes a foundation for the description of the JAE architecture in chapter 3.

2.1 Technology and Terminology

In computer science, the term “agent” is widely used. Preferably, help systems in software applications are described by the term agent (or assistant), but many daemon processes in operating systems and applications, e.g. in network management, are denoted agents as well. The Oxford Dictionary contains following definition for the linguistic use [Ox74]:

> *“person who does something; thing producing an effect; one who acts on behalf of another”*.

In general, the definition of the AI-agent coincides with the description given above; it is only mapped to computer science. In [RuNo95] it is said *“a [software] agent is anything that can be viewed as perceiving its environment and acting upon that environment [...] through encoded bit strings. [...] Its goal is to do a good job of acting on that environment”*. Pattie Maes has added a crucial attribute to this definition, describing an agent more precisely as an autonomous agent: *“[...] by doing so [autonomous agents] realize a set of goals or tasks for which they are designed”* [Mae95]. Thus, an agent is software that can be regarded as an assistant that acts autonomously on behalf of the human being. But this description still allows various interpretations, resulting in many new terms: *network agent, itinerant agent, personal agent, web agent, interface agent, news agent, user agent*, etc.

To overcome this wide variety and to better categorize the agent types, different proposals for taxonomies and attributes have been defined, but also examined critically. Very demanding classifications with focus on sociological aspects of software agents cover only a little part of the agent world [Pet97]. Other proposals, however, have a more general approach, but there are also positions casting doubt on these definitions [Wag98]. A less restrictive view on agent’s properties allows a more useful classification. The universally applicable and often quoted attributes for the characterization of agents are listed in Table 2-1.

Property	**Meaning**
1. reactive	perceives and reacts to changes in the environment
2. autonomous	exercises control over its own actions
3. goal-oriented	manipulates the environment on purpose to achieve a goal
4. temporally continuous	is a continuously running process
5. communicative	communicates with other agents and people
6. learning	changes its behaviour based on previous experiences
7. mobile	able to transport itself from one host to another
8. flexible	actions are not scripted
9. believable	embodies an own believable “character” and emotional state

Table 2-1: List of Agent Properties

Intelligent agents should satisfy the first four properties per se. All other agents can be classified according to the subset of the listed properties, resulting in a hierarchical classification based on set inclusion; for example, mobile and communicative agents are a subclass of mobile agents. A detailed examination of agent definition and classification has been published in [FrGr96]. Taking such properties as flexible and learning into account, the authors sum up in the following definition: *"An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."*

Having this definition in mind, it is desired to further divide the multitude of possible agent classes into an agent taxonomy that is separated into two main classes, namely, *Single Agent System* and *Multi Agent System.* Both classes are subdivided into two subclasses, allowing a coarse categorization of agents into four classes that are denoted as: *stand-alone agents, network agents, inter-linked agents*, and *mobile agents*. The attributes intelligent and autonomous may apply to all four classes of agent types. Table 2-2 shows a revised and expanded version of the agent taxonomy proposed in [MRK96] and enumerates some example applications.

	Single Agent System		Multi Agent System	
	stand-alone agents	network agents	inter-linked agents	mobile agents
applications	Personal Assistants, Advisory Assistants, Meeting Scheduler, User Interfaces, Spell Checker, etc.	Personal Assistants, Smart Mailboxes, Information Access and Management, Web Spider etc.	Distributed Problem Solving, System and Network Management, Electronic Market, Workflow Management, etc.	Telecommunication and Network Management, Internet services, Electronic Market, etc.
communicative	no	no	yes	yes
mobile	no	no	no	yes
intelligent	properties defining intelligent agents as well as the attributes learning, flexible, and believable may apply to all classes			

Table 2-2: Agent Taxonomy

This arranging of agent types makes clear that intelligent and mobile agent technology are not mutually exclusive. Quite the opposite, the AI-based definition of an agent becomes more and more important with the increasing interest in developing mobile agent-based applications.

Applications based on stand-alone agents, often called personal assistants, have been advanced from the vision of a virtual secretary and are already common in smart email systems, word processing applications, scheduler programs, and many more applications. They are individual processes liberating users from routine tasks. Many of these personal assistants have network connections and e.g. take over the task of searching for information in the WWW (*just-in-time information retrieval agents - JITIR agents*, [RhMa00]), co-operate to provide personalized services for mobile users over the Internet (*Personal Mobility Management System - PMMS*,

[HAK03]), or looking after appointments of the user (*RETSINA Calendar Agent*, [PSS02]). These agents can be categorized as network agents corresponding to the taxonomy given above. The *Foundation for Intelligent and Physical Agents* (FIPA, [FIPA02]) try hard to standardize the agent technology, considering personal assistants. First experiences based on the FIPA agent technology have been gained with the personal travel application implemented during the European Union sponsored ACTS projects and are described in [SNB+00].

Inter-linked agents need to have the ability to communicate to jointly solve problems and co-operate. Therefore, interlingua languages have been proposed that guarantee corresponding agents on the same basis of semantics. The most wide-spread and well-known *Agent Communication Language* (ACL) has been proposed by researchers of the DARPA[1] Knowledge Sharing Initiative and is the combination of a vocabulary, an inner language called *Knowledge Interchange Format* (KIF, [GeFi92]), and an outer language called the *Knowledge Query and Manipulation Language* (KQML, [FFMM94]). Based on speech act theory which states that individual communications can be reduced to a small number of primitive *speech* or *communicative acts*, a message is composed as follows: a KQML expression contains parameters, i.e. requests and replies, that are terms or sentences formatted in KIF and expressed with a predetermined vocabulary. The widely used KQML has also influenced FIPA's standard ACL [GIOP02]. Usually, these inter-linked agents work distributed and are co-operative deriving advantages of the ACL, which is used on any transport protocol.

Many of the *Distributed Artificial Intelligence* (DAI) Multi-Agent Systems communicate by means of a virtual (distributed) memory, called *blackboard system*. The principle of a blackboard is simple: it can be considered as a database to which information can be posted that might be of interest to solve a requested problem. By collecting all the information, parts of the problem can be solved and thus can iteratively lead to an overall result. Examples for distributed problem solving systems are speech recognition applications, such as *HEARSAY II* [EHLR80], the system *M* integrating multiple reasoning agents to create a personal assistant [Rie94], the concurrent engineering system *SNEAKERS* demonstrating planning and controlling of tasks [Dou98], or just for the safe coordination of agents in a multiagent system [MMU01].

Certainly, there are just as many applications based on Multi-Agent Systems supporting communication by means of message passing. Researchers in *Computer Supported Cooperative Work* (CSCW) see potential in Multi-Agent Systems deploying agents to perform low-level functions such as task allocation, scheduling, and triggering [LZCH02]. In a similar way, agents are of interest in work-flow management; they are to control and adapt to local situations in a work-flow. The agents are differentiated as dispatching and coordinating agents called worklets which ensure that global constraints are not violated, while maintaining global efficiency by dynamic adapting to a variety of other software processes - such as configuration

1. Defense Advanced Research Projects Agency

management, deployment, validation and evolution [VKK01]. The classical approach to make use of mobile agents in electronic markets already has been made by General Magic with Telescript in 1994. Today, the cyber-market based on mobile agent technology has to cope with the vulnerability caused by a mobile computing environment [KiNo03]. A multitude of research activities are also ongoing in the area of network management. Mixed strategies with interlinked and mobile agents have been investigated (see also chapter 5). The *Perpetuum Mobile Procura* research group of Carleton University also has a special focus in this area and has made many research efforts investigating mobile agent technology in network management [WPB99].

2.1.1 Mobile Agent Principles

Having briefly introduced the agents' world in general, a more precise definition of a mobile agent is necessary. In the following, mobile agent technology is understood as an enhancement of distributed object computing. Before considering details of mobile agents, the traditional and widespread *Remote Procedure Call* (RPC, [RPC97]) in distributed systems is described to have a foundation for comparison.

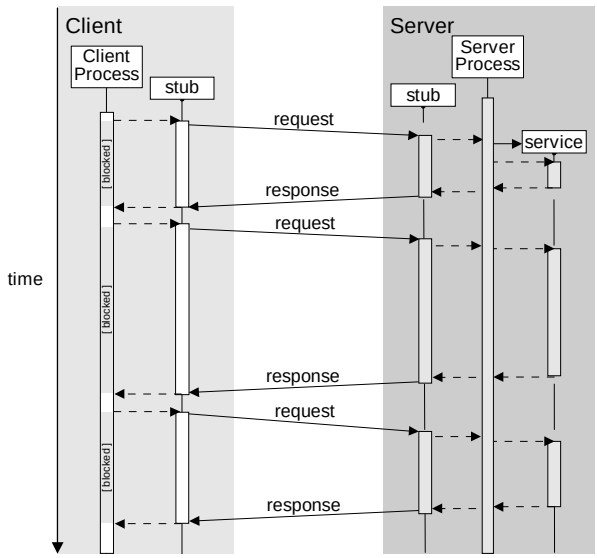

Figure 2-3: Sequence of Traditional Remote Procedure Calls

RPC allows a program running on one host to cause code to be executed on another host. The protocol is initiated by the client and intends to have the client and server processes to communicate by means of their stubs which realize the set of rules for marshalling and un-marshalling parameters and results. The marshalled parameters and results are sent over the network. Besides the set of rules for encoding and decoding information transmitted between two processes, a few primitive operations are defined to invoke an individual call, to return its results, and to cancel it. Furthermore, the operating system ID is given as well as a process structure to

maintain and reference the state that is shared by the participating processes. A sequence of traditional RPCs is illustrated as a sequence diagram in figure 2-3. It shows that the calling process is suspended while the call is in progress and is resumed when the procedure terminates. The remote procedure itself may call other remote procedures that can be located anywhere in the distributed system.

Unfortunately, there are many variations and subtleties in various implementations, resulting in a variety of different and incompatible RPC protocols. Therefore, the marshalling component and the encoding component can be brought together by the *Interface Definition Language* (IDL). The IDL provides the standard interface between objects, and is the basic mechanism for object interaction. It defines the signatures of RPC operations, which contain the name of the operation, its input and output parameters, the results it returns, and the exceptions it may throw.

A mobile agent is defined as an object with a private thread of execution, which is also known as an active object. This active object contains a fragment of code and its execution state, and it is able to migrate in a network of computers. In distributed systems, mobile agents are considered as an alternative or enhancement to client/server technology [PSW96], enriching distributed applications by continuously running processes and code mobility. Thus, the distinctive feature of mobile agents is that they are not bound to the original host where they start their execution. They have the unique ability to transport themselves from one host in a network to another. They move to the host containing the object they intend to interact with, thus taking advantage of being at the same host as the object and thus exploiting the benefits of local communication. In other words, mobile agents can migrate to the resources they need, benefit from

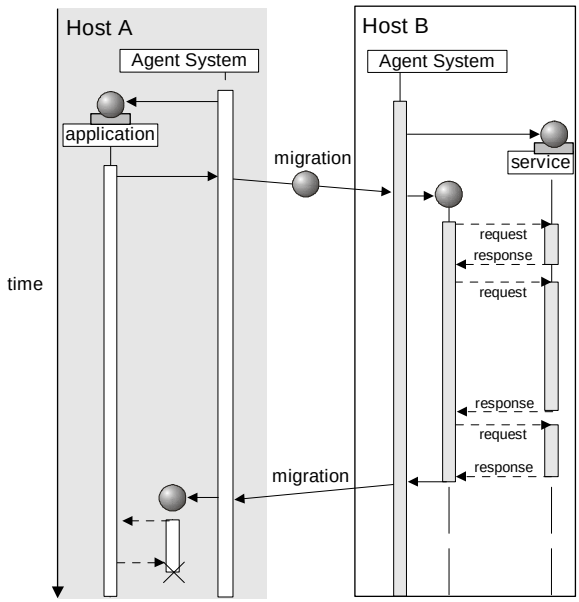

Figure 2-4: Remote Programming

the local high-speed communication, and unblock the initiating application by means of their asynchronous work. It is the mobile agent that decides where and when to migrate and there is no other control instance.

Maybe this is also the reason why mobile agents appears to be autonomous because they fulfil a task without any further interactions with the initiating application or user. Fundamental for mobile agents is a suitable network of hosts allowing them to move from one physical point to another within the distributed system. Agent systems on each of these hosts provide the necessary, basic infrastructure for the agents to migrate, to communicate, and to access system resources and other applications. Thus, they set up the distributed *(mobile) agent network* by supplying the *remote programming* [Whi96] concept. Figure 2-4 depicts the remote programming concept described so far.

The migration process of mobile agents differs fundamentally from RPC. As indicated above, for each migration, it is mandatory to save all current data states, the last execution state, the results, and the program code of the agent. Depending on the underlying agent system, two alternative methods are distinguished: *strong* and *weak migration* [GhVi97]. A strong migration is given if access to the execution environment is provided so that the actual data state and pointers to the execution state can be obtained. In this case, a complete location transparency for the application is maintained. After the migration to the new host, the mobile agent continues exactly at the last execution state. Therefore, the whole execution state of the agent, the registers and results in memory must be stored, transferred to the destination, restored, and executed at the new host.

Object migration, in particular with transparent control of all co-operating threads, is a very lavish and difficult task. Persistency and object migration has been subject to research, e.g. in the *Tycoon* Project. Very detailed descriptions of the virtual machine, object as well as thread migration combined with persistency considerations can be found in [Mat96]. Another example for strong migration is given with *Telescript* or the mobile agent system *ARA*, which supports TCL, allowing mobile agents to continuously execute instructions while changing the hosts [Pei02]. Most of the higher level languages used as implementation language of mobile agent systems are interpreted and executed on virtual machines, which do not permit the access to their registers and memory stacks. The upshot of this is that the program execution state cannot be determined and the agent programmer has to decide, with the help of additional location or state information, which program code to execute after a migration.

By now, most of the mobile agent systems are based on this weak migration, although different solutions and techniques are used. For example Mitsubishi's *Concordia* agents [Con97] carry a list determining which methods to execute at which location. In *Voyager* [Whe03] (which is not an explicit agent system but an agent enhanced *Object Request Broker* (ORB)) or the FIPA compliant agent framework *BOND* [Böl02], the method to be executed at the destination host is determined with the migration command: e.g. *moveTo(destination host, method)*. Most com-

mon is the use of a method which returns a unique information executed at each restart of the agent. This information can be the number of migration steps or the name of the current agent system. Depending on this data, further decisions are made by the mobile agent. The first Java agent system implementing this agent migration strategy has been the Mole system [BHR+98].

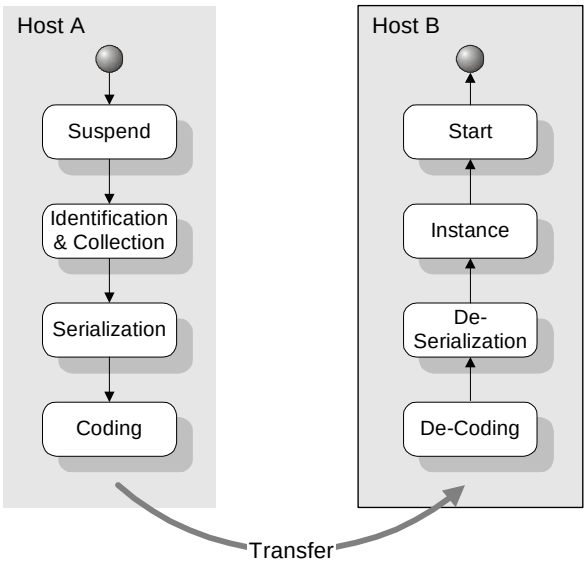

Figure 2-5: Migration Process Steps

As shown in figure 2-5, the migration process of a mobile agent can be subdivided into 9 steps. In both the weak and the strong migration, the object stored in the memory of the computer has to be converted into a transportable form. This is done by serialization of the agent object's current state into a sequence of bytes. This serialized byte-stream is formatted according to a transport protocol and sent to the destination, where the object is restored vice versa. In detail, the agent migration process is divided into following steps:

1. **Suspension of the agent**: The agent execution is stopped and the current state of this object is captured. Depending on weak or strong migration, the content of registers/memory and the process status are stored.

2. **Identification & Collection**: All relevant objects of an agent have to be identified and collected for the next step. If dynamic reload of objects over the network is supported (as in Java), only belonging data is stored and the objects' locations are transmitted with the mobile agent.

3. **Serialization**: Serialization of a class' instance is needed to restore an object or to keep objects persistent. Therefore, the complete state of an object including any object it refers to (e.g. values of all instance variables) is stored in a byte stream and is unambiguously marked.

4. **Coding**: Depending on the used transfer protocol, the classes and serialized objects have to be coded and signed for physical transmission.

5. **Transfer**: Preceding the transmission of objects, a protocol guarantees the authentication and the secure transfer of the serialized objects.

6. **Decoding**: At the destination, the objects are decrypted and decoded. Thus, the serialized form of the objects is accessible.

7. **Deserialization**: Out of the sequence of bytes, the object code, its state and its instance variables are restored and kept in memory.

8. **Instance**: The instance of the mobile object is reconstructed with the help of the object code and its instance variables. It should have the same execution state as before its transmission.

9. **Execution**: The mobile agent is executed. Depending on strong or weak migration, the agent continues with the next execution state of the program pointer or the control mechanism supporting the weak migration.

Handling of mobile objects	System/Technology
Delegation/Remote Execution	Servlets, CORBA [COR02]
Code on Demand	ActiveX, Java Applets, Javascript
Weak Migration	Mole, IBM Aglets [LaOs98], JAE
Strong Migration	ARA, Tycoon, Telescript, AgentTCL [KGN+97]

Table 2-6: Applications of Mobile Objects

Mobile objects have gained momentum by the growing interest in the remote programming paradigm, but it is not the only field of application. Alternative performing of mobile objects as listed in Table 2-6 have been prominent before. Further detailed examination of agent migration concepts and mobile code can be found e.g. in [HKB97, Fün98, MiRa00, Sat01].

2.1.2 Terms and Definitions

Having introduced remote programming and migration as principles of mobile agents, the main components of mobile agent technology will be defined and described to deliver a common basis of the terminology used in the remainder of this work.

Agent System

The key element of mobile agent technology is the agent system. Further common names are agent engine, agent server, agent host, or agent platform, all of them mentioned here merely for the sake of completeness. In the remainder, only the term agent system will be used. Such an agent system provides the infrastructure in which agents can be deployed on a host. It allows all facilities to start, register, administer, and terminate agents. Furthermore, it provides the

core technology for communication, mobility, security, and services. Each of these technologies is encapsulated in one or more components of the agent system. The agent system is started as a resistant process on each host that is willing to provide access for mobile agents and provides not only the execution facility, but also services and libraries. An agent system on top of an abstract machine can be seen as a middleware, which is an intermediate layer between the applications and the underlying heterogeneous networks and operating systems. Figure 2-7 depicts the architecture of an agent-based middleware.

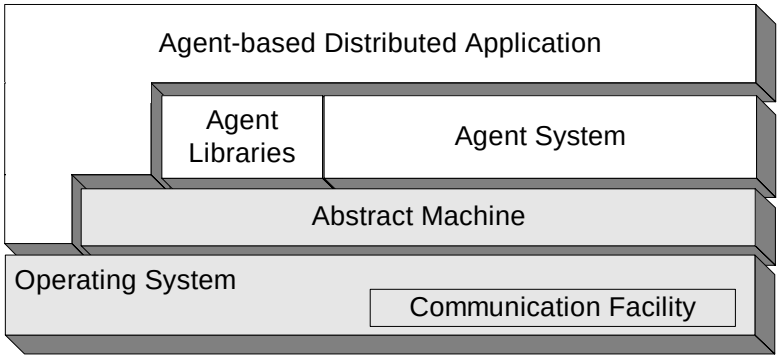

Figure 2-7: Coarse Architecture of an Agent-based Application and its Middleware

This middleware releases applications based on agents from platform dependencies and reduces the complexity of connections between heterogeneous physical environments. The coarse architecture further clarifies the resource access of agent-based applications. For system agents (see next paragraph), the direct use of the operating system and the abstract machine resources are permitted, for mobile agents it is not. The set of shared resources and services within the agent system enables co-ordinated, to some extent distributed, applications use network and operating system resources. A multitude of these agent systems establish an *agent network.*

Mobile and System Agent

Agents consist of program code, user data, system data with current state information, and a unique ID. The program determines the behaviour of the agent. The user data contains values of local variables and data objects that can be documents, any other files, or electronic cash of the agent. The unique ID guarantees the identification of each agent in the global agent network. Further system information of interest that mobile agents may carry with them are host of origin, number of hops (location changes), name of the owner, membership of a group (collaboration), authentication, and permission keys.

Although both mobile and system agents may be derived from the same agent class, there is one essential difference: system agents are stationary[1], i.e. they do not move between different agent systems, and have nearly unrestricted access to local resources. Usually, they offer serv-

1. the term stationary agent is often used synonymously with system agent

ices to mobile agents. Services can be e.g. clients for database accesses, mail servers, or any other non-agent applications such as the *Common Object Request Broker Architecture* services (CORBA, [COR02]). They can consist of simple classes, but are not restricted to them. On the one hand, a service can be a full package with many internal objects. On the other hand, such a *service agent* can be a simple *wrapper* to some legacy application interfaces. Decisive for mobile agents is security, because they are executed in various agent systems and hence, cannot be considered trustworthy. Once a mobile agent enters an agent system, it can only process its internal data and call services controlled by the agent system. Interaction with other agents or the usage of system/host resources can only be achieved through system agents or the agent system. The general life cycles of a mobile agent is shown in figure 2-8.

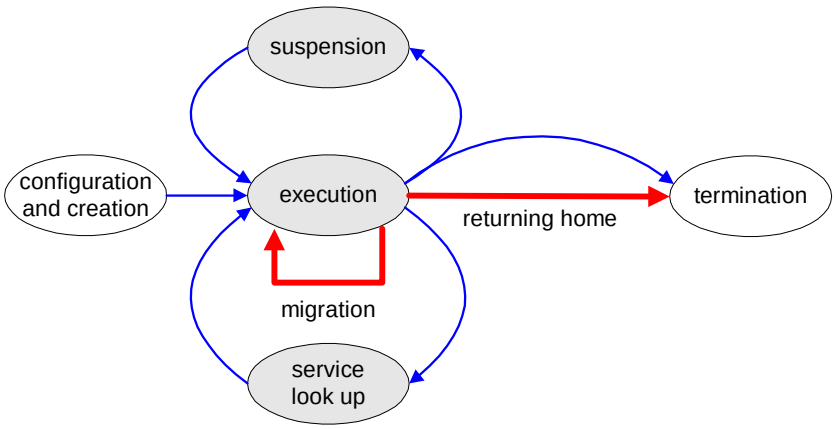

Figure 2-8: Life Cycle of a Mobile Agent

After configuration and creation, an agent executes its code and may switch to suspend state or the service look up state, which are both supported by the agent system. During execution, mobile agents can change their physical location through migration. They can terminate at any host, except if results have to be returned to the origin.

Agent Framework

Development tools and methods play an important role in application programming using agent technology. All components necessary for testing, debugging, validation, visualisation, and simulation belong to the agent development environment. The agent framework consists of both the development environment and the agent system. In general, the agent framework sticks to one programming language. Only in research, considerations of multi language support are made [Pei02]. At present, there are no commercial agent frameworks available which provide interoperability of agents from different agent system providers. Agent-based programming is in its beginnings, but interoperability across the life cycle of agent tools and different manufacturers is indispensable. Here, standardisation becomes more and more inevitable, although the nature of mobile agent makes it very difficult to comprehend the concurrent and non-deterministic activity of mobile agents.

Agent Language

The programming language which the agent system and the agents are implemented in is described as the agent language. To overcome the heterogeneity of the different machines in a distributed system, interpreted languages or languages supporting an intermediate format in combination with an abstract machine are best choice. The prevailing programming language for mobile agents is the platform independent language, Java.

The basic idea of Java and today's open systems is the same: applications written once are to be executable on heterogeneous platforms without modification. Sun Microsystems' solution interprets the idea more topical and comprehensive and meets the requirements of open systems - even causing current systems to be dispensable. Java is a software concept that provides an object-oriented language as well as its execution environment. Primitive data types are rigorously specified and not dependent on the underlying processor or operating system. Furthermore, these data types allow secure execution of objects by eliminating the conventional pointer model as known e.g. in C and preventing the possibility of over-writing memory and corrupting data. The security architecture of Java makes it reasonably safe to host mobile agents, because agents cannot tamper with the host or access private information.

In the terminology of Java, the execution environment (abstract machine) is called the virtual machine and offers well-defined interfaces that make applications independent not only from hardware, but also from operating systems. Dynamic class loading allows the virtual machine to load and define classes at run-time, which is a considerable advantage for an agent language with mobile objects. Multi-thread programming is supported as well as a set of synchronization primitives that are built into the language. A key feature of mobile agents is that they can be serialized and de-serialized. Java conveniently provides a built-in serialization mechanism that can represent the state of an object in a serialized form, sufficiently detailed for the object to be reconstructed later. Many more useful libraries such as network and graphic user interface packages have helped this language to become widely spread and popular. Solutions based on Java become scalable and data remain useable without difficulties in adaptation. Optimization of the Java Virtual Machine (JVM, [LiYe99]), according to the underlying operating system, is sufficient. Some pioneers in mobile agent technology predict mobile Java agents will become ubiquitous in distributed object computing within the next few years.

The *Agent Communication Language* (ACL) is the basic element for inter-agent communication, which is based on speech act theory. In AI, they are the most important aspect in multi agent systems. Today, derivatives of KQML and KIF gain importance, particularly in standardization bodies.

Agent Transfer Protocol

Essential for agent migration is the *Agent Transfer Protocol* (ATP). This application protocol, on top of the transport layer, ensures that authentication and transfer of agent state and code are

performed correctly. Depending on requirements of the agent system, it is the task of the ATP to negotiate the resource requests with the destination system and grant permission to mobile agents.

Agent Directory

The *Agent Directory* (AD) is the portal to the agent network. System agents offering a global service export their service descriptions to the AD. The address of the AD is well-known to all agent systems. The AD is a database that contains not only service descriptions, but also stores all agent network relevant data and even registers roaming mobile agents. It is a key element in an agent infrastructure and plays an important role for autonomous actions of mobile agents.

2.2 Why Mobile Agents?

None of the distributed software applications and solutions are bound to mobile agent technology. Mobile agents are merely design "patterns" for software. Object oriented technology can be used to enable agent-based technology. The advantage of an agent framework in comparison to a plain object oriented framework arises since the mobile agent paradigm is not yet supported in object oriented technology. Consequently, agent-based tools preferably have the agent design patterns inherent in their framework. Otherwise, the programmer has to explicitly program the properties which come with mobile agent technology. In other words, the most important software engineering argument is: each individual property of mobile agent technology (reactive, autonomic, goal-oriented, etc.) can be addressed in some (ad hoc) manner without mobile agents, whereas a mobile agent framework addresses all of them at once [ThTh97, BCS01].

Fundamental to this technology is that agents offer a way to overcome heterogeneous networks with incompatible programming interfaces, data formats, and protocols. In the Agent Technology green paper of the *Object Management Group* (OMG), it is simply said: *"agent technology provides the ultimate in distributed component technology"* [OMG00]. The most beneficial reasons for mobile agent technology are:

- *Network load reduction and efficient resource exploitation*: if an agent's goals require extensive communication with a particular resource in the network, there are issues of traffic and bandwidth to consider. Furthermore, a mobile agent might have better access to remote data than services offered at another site. In either or both of these cases, moving to the resource could be more efficient and reduce or even eliminate network traffic, allowing the agent to perform its duties more quickly. Certainly, one disadvantage of such a mobility is that the remote host must provide the processing power to support the mobile agent's execution. Mobility not only brings the burden of additional

processing of the CPU, but raises issues of security, billing the agent for its resource usage, and unforeseeable scalability problems. Nevertheless, mobility is an important property for agent-based distributed applications, which has many advantages for certain classes of applications.

- *Distributed parallel and dynamic processing*: mobile agents possess the unique ability to clone themselves and/or use other agents. They distribute themselves or other agents among the hosts in the agent network in order to maintain an advantageous configuration for solving a particular problem: for example, a main task can be broken into smaller sub-tasks and it can then be distributed for parallel execution.
- *Disconnected and autonomous operations*: mobility and autonomy allow an agent to move from its point of origin into a network and continue to operate, even if the originating device is temporarily or permanently disconnected from the network. Thus, agents can provide services and satisfy pre-defined goals without user intervention. Even critical real-time systems such as robots in manufacturing processes can benefit from this property. Dispatched from the central controller, mobile agents can respond to changes in their environment and act locally corresponding to the initial controller's directions. In Telecommunication, agents gain importance because of their ability to provide disconnected and personalized user support.
- *Simple software update*: Mobility makes software updates and maintenances easy. Any agent-based application can be updated during execution, simply by distributing a new version of the mobile agent. Particularly in distributed systems, the update of interfaces, data formats, or protocols, which evolve to accommodate new efficiency or security requirements, is a cumbersome if not impossible task. Mobile agents are able to move to affected hosts in order to establish a new interface, convert data formats, or setup communication links based on proprietary, new protocols.
- *Seamless and flexible integration of services and heterogeneous hosts*: a distributed system is basically heterogeneous, often from both hardware and software perspectives. Agent systems are considered as middleware and thus cope with computer- and transport-layer-independency, which is a good foundation for seamless system and service integration. The ability of mobile agents to react dynamically to unfavourable situations and events makes it even easier to support robust and fault-tolerant distributed applications. For example, if a host with all its services is being shut down, all agents executing on that machine will be warned and given time to react. They may e.g. continue their operation on another host in the network. Extra services and hosts can be flexibly added to or removed from the network, maintaining better work load in the agent network.

Until today, none of the agent systems accomplishes aforementioned properties satisfactorily. The main deficiencies result from the underlying and wide spread JVM. Pending research issues to be solved are e.g. control mechanisms for thread migration and resource consump-

tion. The risk of denial of service attacks is quite high, because Java does not provide means to the host to limit an object's allocation of processor and memory resources. However, the ability of a object to allocate external resources - for example by opening files and sockets or creating windows - has been reduced with version 1.2 of Java. Support for preservation and resumption of the execution state is still missing, which is necessary for strong migration and full transparency of object distribution. However, for security reasons the retrieval of an object's full execution state is impossible in Java. Information such as the status of the program counter and frame stack will remain a permanently forbidden territory for Java programs. Finally, in spite of the outstanding type concept of the language, there is the drawback in reference protection. Java objects' public methods are accessible to any other object that has a reference to it. There is no programming interface provided so that an agent/object can monitor, in order to control which other agents/objects are accessing its methods.

2.2.1 Mobile Agent Applications

Mobile agent technology makes application programmers' work easier. As mentioned earlier, it is understood as a design pattern, especially for development of distributed software. Agent technology comes along with a set of convenient tools and libraries, allowing very fast designing and developing prototype of distributed applications. Advantages in fast application development cannot be described in numbers and benchmarks; it is more a practical experience that programmers have to experiment with and understand. The inventors of IBM's Aglet have recognized this fact and have gone a step further, proposing a design pattern classification in order to enclose the area of applications and try to define reusable patterns for agent-based applications.

They believe in agent design patterns that can help capturing good solutions to common agent design problems. Roughly, they identify three main classes: travelling, task, and interaction patterns. An application developer can choose from a set of travelling patterns dealing with agents' itinerary, permission, auto forwarding, and matters of Quality of Services (QoS). The task patterns address the distribution of agents' work. They distinguish between the master-slave and the plan pattern. The former allows software solutions with parallel execution of delegated tasks that are fulfilled by so-called "slave" agents. The latter pattern provides a way of defining the co-ordination of multiple tasks that are to be performed on multiple hosts. Finally, the interaction patterns support the programmer with agent solutions for local communication at a given host (meeting), persistency service for mobile agents (locker), remote message delivery (messenger), groups of agents that travel and act together (group), and methods for naming and locating of agents with specific capabilities (facilitator) [ArLa98]. Although the classification seems to fit merely to their agent system, the idea of agent design patterns is certainly an interesting research task and could enrich agent frameworks.

In summary, there are no specific “mobile agent applications”, but there are plenty of application areas that highly benefit from using mobile agent technology (see Table 2-2). The number of research projects and prototype implementations dealing with mobile agent technology clarify the potential behind this technology. In IBM publications, eight application areas have been identified that agent technology can enhance advantageously [JGA+96]:

1. system and network management,
2. mobile access/management,
3. mail and messaging,
4. information access/management,
5. collaboration,
6. workflow and administrative management,
7. electronic commerce,
8. and adaptive user interfaces.

The agent’s property of mobility, however, is an interesting and distinguishing feature, which has introduced a new programming paradigm to the design and architecture of agent-based applications. Besides the aforementioned range of applications, mobile agents have been recognized as a promising technology for telecommunications [ICM98, PLKS99, MaGl02]. Mobile cellular systems (e.g. GSM), *Intelligent Networks* (IN, [MaPZ96]), or *Telecommunications Management Networks* (TMN) can benefit from mobile agent technology [IEEE98, CMV+00].

In Europe, the merging of telecommunications and agent technology has intensively been investigated in several research projects. The *Cluster for Intelligent Mobile Agents for Telecommunication Environments* (CLIMATE, [CLI98]) has tried to protocol the activities in each of the international projects. It represented a pool of projects within the European Union collaborative research and development programs on *Advanced Communications Technologies and Services* (ACTS). Mobile agent technology at the boundary between the fixed and wireless network is also a main topic of this thesis and will be discussed in detail in chapter 4.

2.3 Outline of Agent Standards

The demand for a standardized agent systems has quickly arose. The wide variety of proprietary agent systems has created many new ideas in this research field, but co-operation and network-wide communication as well as migration of mobile agents has not been realized yet. Even agent systems implemented in the same programming language cannot be brought together. The desire to allow mobile agents of different agent system manufacturers to co-operate and communicate has lead to standardization efforts. The first notion of an agent organiza-

tion dealing with agents has been put forward by agent industry pioneers, in order to facilitate the emergence of agent technology, its applications, standards, markets, and collaboration. The idea that industry-driven organizations on a global scale could simplify and expedite the distribution of the mobile agent environment has been originally picked up by two organizations: The mission and objectives of the *Agent Society* was to provide a small-scale international industry and a professional umbrella. They have supplied a technology transfer among enterprise, research communities, and individuals by tackling a homepage and an email list. In 1998, the initial merger of individuals and companies has been incorporated as a non-profit corporation in the State of Washington, but in the meanwhile the Agent Society has been dissolved. The second organization is the *Foundation for Intelligent Physical Agents* (FIPA, [FIPA02]) and has been established in 1997. The third standardization body worth mentioning has been originated by the *Object Management Group* (OMG, [OMG]). The primary goal of the OMG Agent Working Group was to integrate agent mobility into the CORBA standard. Since the first draft standards were published in 1997, the organizations have set up liaison agreements to strengthen their collaborative efforts and to encourage agent technology standards. Their purpose is to evolve consistent object technology standards and to further co-ordinate OMG's and FIPA's related work.

2.3.1 Mobile Agent System Interoperability Facility

The efforts of OMG's Agent Working Group to integrate agent mobility into CORBA and to standardize interoperability of agent systems have resulted in the *Mobile Agent System Interoperability Facility* (MASIF, [MASIF98]) specification. In this specification, all MASIF relevant terms are defined and a standard syntax for locating and naming of agents and agent systems is determined. The focus, however, is on agent system architecture, agent management and agent transfer, which are implemented in the same agent language. Only little attention is paid to agent communication. Although the communication needs of mobile agents cannot be covered by current distributed object systems, the only reference is made to *Remote Method Invocation* (RMI), the object communication of CORBA, and the *Distributed Component Object Model* (DCOM) of Microsoft. These object communications are sufficient for stationary agents, but the mobility of mobile agents raises new problems that are not addressed with these communication facilities. In MASIF, communication between agents is supported by the *Communication Interface* (CI), which - besides agent communication - integrates the migration service of agents. As depicted in figure 2-9, the CI lies on top of the transport layer and hides all underlying protocols. In MASIF, the migration process is divided into two steps: *agent transfer* and *class transfer.* The agent transfer matches with the migration process described in figure 2-5. Additionally, a QoS for the migration process is proposed, but without a detailed specification. If the destination agent system grants access and the QoS demands are fulfilled, the migration process is initiated. In contrast to the migration process described before, not all necessary classes are transferred during the first step. With the specification of the class transfer, the MASIF standard supports dynamic class reloading; i.e. missing classes

can be reloaded from a specified code-base during execution. The subdivision of the migration process allows various realizations of agent mobility. The aforementioned migration process is just as realizable as a class transfer with a central code-base in the agent network. In the latter case, each agent system can serve as a class provider holding the central code-base.

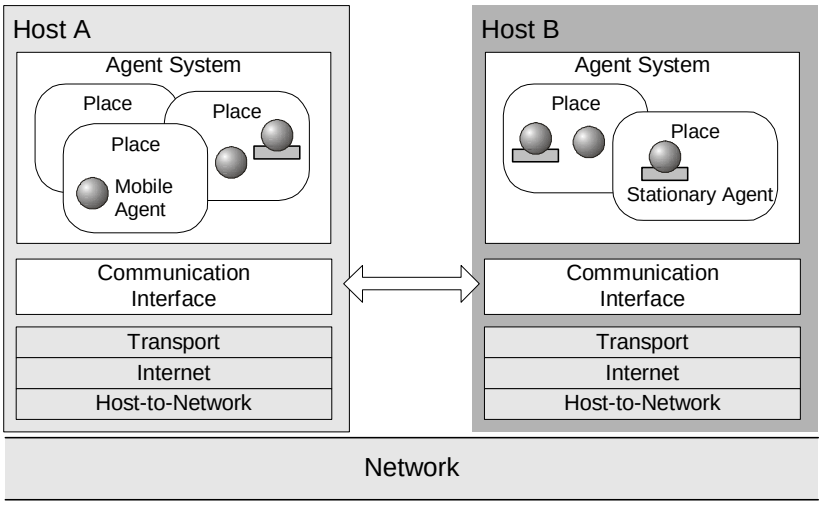

Figure 2-9: MASIF Reference Model

The MASIF reference model places the agent system on top of the CI and relates it unambiguously to one host. The specification does not define any limits concerning the number of agent systems running on the same host. Each agent system contains at least one so-called *Place*, which serves as a location where agents execute and meet. The idea behind places is to assign different permissions to agents depending on the execution place. The interoperability of mobile agents is ensured by the interfaces of the *MAFAgentSystem* and the *MAFFinder.* The former provides a common *Application Programming Interface* (API) for agent creation, agent transfer, class transfer, and the management of agents and places. The *MAFFinder* provides naming services for agents, places, and agent systems. Accordingly, the standard distinguishes different authorities which have to be authenticated in the agent network. An agent authority is an individual or an organization that executes an agent or an agent system. In case of multiple agent systems of one authority, this agent network is named *Region.* Figure 2-10 depicts the Region concept of MASIF.

The idea of Regions is to increase scalability of agent-based applications dynamically and transparently over multiple MASIF compliant agent systems. A Region is a closed agent network with full interoperability of systems and agents. Access to such a Region is given by a well-defined *Region Access Point* (RAP), which also grants non-agent-based applications admission. Furthermore, the RAPs enable the inter-link between different Regions and non-MASIF networks. Correspondingly, the RAP can be compared with a *Firewall* in the Internet. The MASIF standard says that there is only one MAFFinder necessary per Region which, however, can be accessed by all other systems external to this Region. To keep an unambigu-

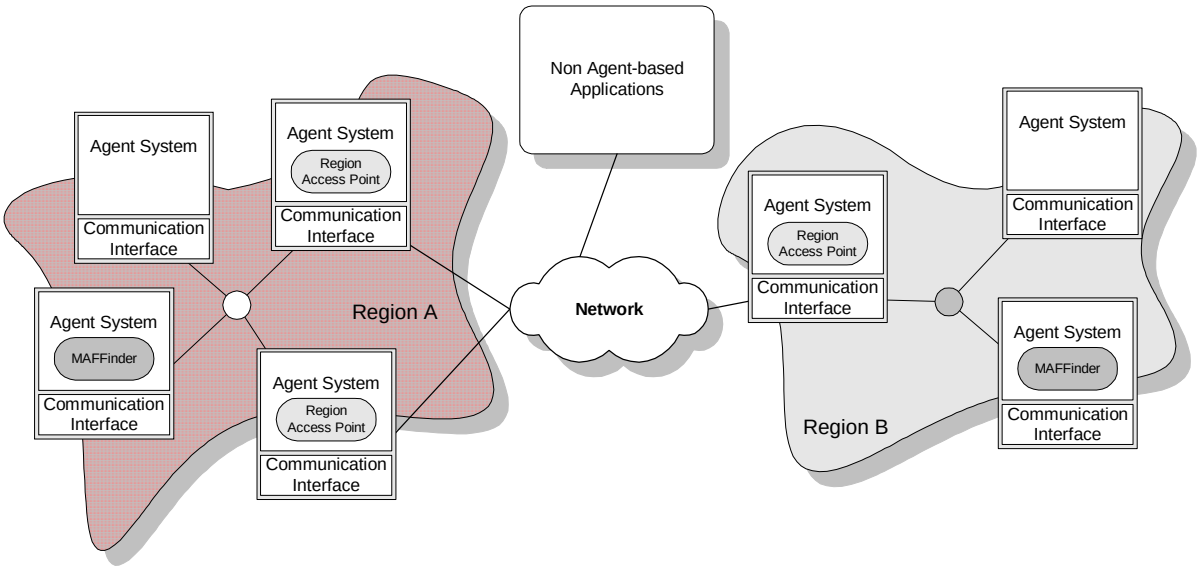

Figure 2-10: Network of Regions in MASIF

ous agent ID throughout the network of different Regions, the ID of a mobile agent is composed of the authority's name plus the unique Region number of the agent.

2.3.2 Foundation for Intelligent Physical Agents

The *Foundation for Intelligent Physical Agents* (FIPA) is an international, non-commercial organization. They promote the development technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings. In the meantime, FIPA's work has become the focus of attention for agent technology. Subject of interest for standardization are aspects such as a common language and techniques for managing individual agents. Until today, the specifications have been rearranged into 5 main areas, the Applications, the Abstract Architecture, the Agent Communication, the Agent Management, and the Agent Message Transport. Table 2-11 gives an overview of all specifications sorted by the topic; note that only specifications that have the standard status are listed. The full list of public papers can be retrieved from [FIPA02].

The application specifications enumerate example application areas in which FIPA agents can be deployed. The specification defines ontology and service descriptions including message transport Quality of Service for nomadic applications and devices. The abstract architecture specifications defines entities that are required to build agent services and the FIPA agent environment. A bigger part is covered by the agent communication specifications which define the Agent Communication Language (ACL) messages, message exchange interaction protocols, speech act theory-based communicative acts and content language representations. The Interaction Protocols deal with pre-agreed message exchange protocols for ACL messages; different utterances for ACL messages are handled by the Communicative Act specifications and different representations of the content of ACL messages are defined in the Content Language

Topic	Specification Part	Version
Applications	Nomadic Application Support	FIPA 02 V. H
	Quality of Service	FIPA 02 V. A
Abstract Architecture	Abstract Architecture	FIPA 02 V. L
Agent Communication	ACL Message Structure	FIPA 02 V. G
- Interaction Protocols	Request Interaction Protocol	FIPA 02 V. H
	Query Interaction Protocol	FIPA 02 V. H
	Request When Interaction Protocol	FIPA 02 V. H
	Contract Net Interaction Protocol	FIPA 02 V. H
	Iterated Contract Net Interaction Protocol	FIPA 02 V. H
	Brokering Interaction Protocol	FIPA 02 V. H
	Recruiting Interaction Protocol	FIPA 02 V. H
	Subscribe Interaction Protocol	FIPA 02 V. H
	Propose Interaction Protocol	FIPA 02 V. H
- Communicative Acts	Communicative Act Library	FIPA 02 V. J
- Content Language	Semantic Language (SL) Content Language	FIPA 02 V. I
Agent Management	Agent Management	FIPA 02 V. J
Agent Message Transport	Agent Message Transport Service	FIPA 02 V. F
- ACL Representations	ACL Message Representation in Bit	FIPA 02 V. G
	ACL Message Representation in String	FIPA 02 V. I
	ACL Message Representation in XML Agent	FIPA 02 V. E
- Envelope Representation	Agent Message Transport Envelope Representation in XML	FIPA 02 V. J
	Agent Message Transport Envelope Representation in Bit Efficient Specification	FIPA 02 V. D
- Transport Protocol	Agent Message Transport Protocol for HTTP	FIPA 02 V. F
	Message Transport Protocol for IIOP	FIPA 02 V. G

Table 2-11: FIPA Specifications sorted by Topics

Specifications. The control and management of agents within and across agent platforms are specified in the Agent Management specifications. Finally, the Agent Message Transport specifications deal with the transport and representation of messages across different network transport protocols, including wireline and wireless environments. Different representation forms for ACL messages and envelopes are defines by the ACL Message Representation and the Envelope Representation specifications, respectively. The different network transport protocols for delivering ACL messages are separately handled in the Agent Message Transport Protocol specifications.

In 1999, FIPA created an Architecture Technical Committee to develop an agent reference architecture to guide FIPA in the development of future agent technology standards. The architecture is abstract, using *Unified Modelling Language* (UML) for more precise specifications.

In the same time frame, OMG's Agent Working Group has started an architecture paper (Agent Green Paper [OMG00]) to guide OMG in the development of an agent technology road-map. Currently, the shared membership between OMG and FIPA makes it possible for both specifications to become well aligned and provide a solution for interoperability.

FIPA has never intended to specify regulation for implementation, but developed reference architectures and defined specifications (not implementations) for the main interfaces in the reference architecture. There are 4 mandatory components which constitute a FIPA conform agent system (named *Agent Platform* (AP)). The AP provides the physical infrastructure in which agents can be deployed. The concept of an AP foresees that agents of a particular AP can be located on different hosts and are not bounded to one machine. Components and interactions of the modules are depicted in figure 2-12 and will be described in the following.

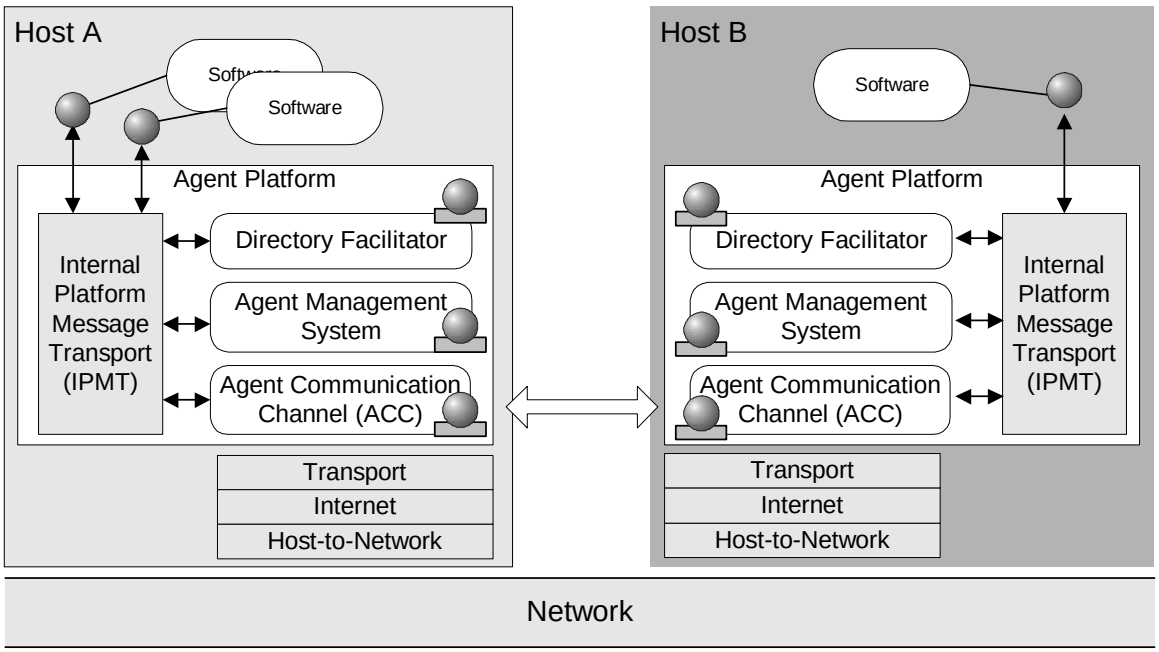

Figure 2-12: FIPA Reference Model

The *Agent Management System* (AMS) is the main control unit of the agent platform. It controls, monitors, and executes all activities within the AP, which includes platform external agent communication and agent migration. Therefore, it maintains a directory of logical agent names and their associated transport addresses for an AP. Tasks of the AMS include creation of agents, deletion of agents, deciding whether an agent can dynamically register with the AP (depending on agent ownership) and overseeing the migration of agents to and from the AP. If the AP executes on multiple machines, the AMS represents the authority across all machines. The AMS itself is an agent just as the *Directory Facilitator* (DF) and the *Agent Communication Channel* (ACC), which are standard system agents of a FIPA agent platform.

The DF serves as yellow pages for services provided by agents. Therefore, agents register their services with the DF or query the DF to find out what services are offered by other agents. FIPA proposes an interesting domain concept that is closely related to the DF, as all registered agents of one DF belong to one logical domain. This logical domain is not bound to one agent platform, but can expand over several systems. On the other hand, each domain has only one DF, which provides a unified, complete, and coherent description of the domain. Thus, the entire agent universe is represented as the set of all domains. Domains may have e.g. organisational, geopolitical, contractual, ontological affiliation, or physical significance. As shown in figure 2-13, an agent may be present in one or more domains and a physical agent platform can define more than one domain having more than one DF agent running on the platform.

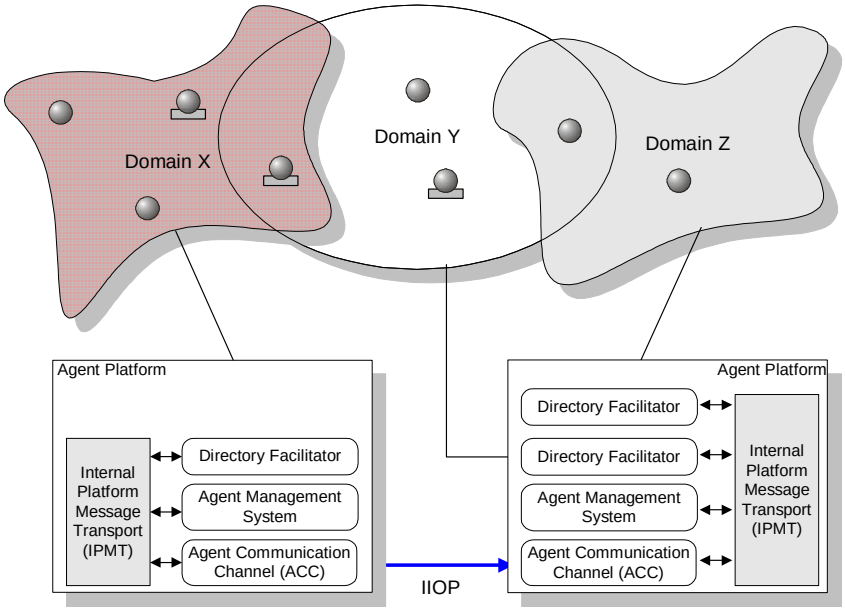

Figure 2-13: Domain Concept of FIPA

The execution of agents is determined by the life-cycle model with the current state being set by the AP. An agent must register with a DF in order to be visible at least in one domain and to exist in this context. Once registered with a DF, the life-cycle model complies with the following 3 rules, which holds for most of the existing agent systems:

1. **Agent system bounded and unique**: An agent is physically managed within an agent system and bounded by its life-cycle to this specific agent system. In other words, each agent has exactly one agent system life-cycle state at any time, within exactly one agent system.

2. **Application independent**: The life-cycle model is independent from any application. It only defines the states and the transitions of the agent service in its life cycle.

3. **Instance oriented**: The agent described in the life-cycle model is assumed an instance, i.e. an agent has a unique name and is executed independently of other resources.

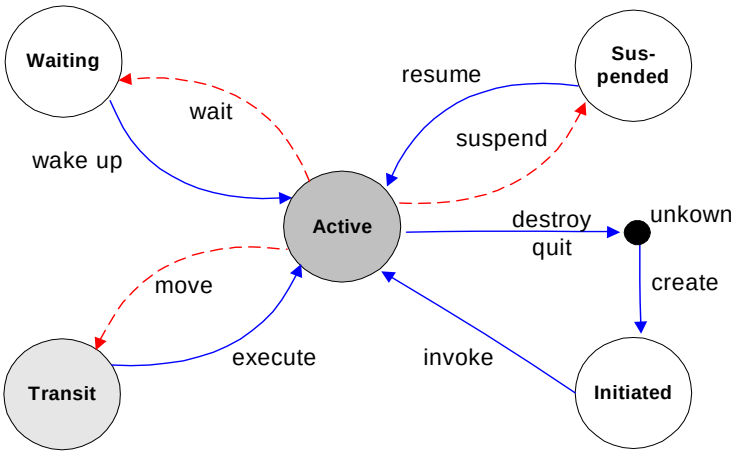

Figure 2-14: Agent Life Cycle of FIPA

The agent's states, shown in figure 2-14, are depicted as circles and the transitions as arrows. Note that actions initiated by an agent system are marked by continuous lines, whereas agent driven actions are represented by dotted lines. Furthermore, the transit state (*move*) can only be entered by mobile agents. This ensures that a system agent executes all of its instructions on the node where it has been invoked. Only the agent system can restart a mobile agent from the transitory state (*execute*). The life-cycle of an agent further includes the following actions: creation/installation of a new agent (*create*), invocation of the agent (*invoke*), ordinary termination of an agent (*quit*, which can be ignored by the agent), agent initiated *wait*, forceful termination of an agent (*destroy*, which can only be initiated by the agent system and cannot be ignored by the agent), suspend (can be initiated by the agent system and the agent itself), resume, and wake up. The proposed agent life-cycle has been adapted into JAE and has been extended by a further state for service look up which will be discussed in the next chapter.

The last mandatory FIPA agent to be described is the ACC agent, which ensures a common communication protocol between agents on the same agent platform as well as on remote agent platforms. All agents have access to at least one ACC. The message routing service offered by the ACC must be reliable and orderly. Routing messages between AP's requires an agreement on a interoperability protocol including transport protocol, encoding and addressing scheme. Although the implementation of the *Internal Platform Message Transport* (IPMT) is not defined explicitly, communication between agent platforms has to be performed with the minimum baseline protocol *Internet Inter Object Request Broker Protocol* (IIOP, [GIOP02]) from the Object Management Group which thus, must be implemented in the ACC.

2.4 Survey of Existing Agent Systems

Since the first appearance of the commercial mobile agent system Telescript in 1994, about half a dozen new commercial systems appeared. Agent technology has clearly gained advantage through the programming language Java which is reflected by the number of systems developed with it. Especially in research the dominance of Java has become apparent with more than 50 systems registered and listed in the mobile agent list maintained by the department of computer science at the University of Stuttgart [MAL00].

Product	Company	Language	Description
AgentBuilder®	Reticular Systems, Inc.	Java	Integrated Agent and Agency Development Environment
AgenTalk	NTT/Ishida	LISP	Multi-agent Coordination Protocols
Agent Building Environment	IBM	C++, Java	Environment
Agent Development Environment	Gensym	Java	Environment
Agentx	International Knowledge Systems	Java	Agent Development Environment
Aglets	IBM Japan	Java	Mobile Agents
Concordia	Mitsubishi Electric	Java	Mobile Agents
DirectIA SDK	MASA	C++	Adaptive Agents
Agent Development Kit (ADK)	Tryllian	Java	Mobile Agents
Grasshopper	IKV++	Java	Mobile Agents
Infosleuth	MCC	Java, Perl, Tk/tcl	Agent Development Tool Agent Library
iGEN	CHI Systems	C/C++	Cognitive Agent Toolkit
Intelligent Agent Factory & Agent Library	Bits & Pixels	Java	Agent Development Tool Agent Library
JACK Intelligent Agents	Agent Oriented Software Pty. Ltd.	JACK Agent Language	Environment
Jumping Beans Engineering	Ad Astra	Java	Mobile Agents
Kafka	Fujitsu	Java	Agent Library
LiveAgent	AgentSoft Ltd.	Java	Internet Agent Construction
Microsoft Agent	Microsoft Corporation	Active X	Interface creatures
Swarm	Swarm Development Group	Objective C, Java	Multiagent simulation
Versatile Intelligent Agents (VIA)	Kinetoscope	Java	Agent Building Blocks
Voyager	Object Space	Java	Agent-Enhanced ORB

Table 2-15: Commercial Agent Systems and Environments

Lessons learned so far are that agent technology is not a single, new, emerging technology, but an integrated application of multiple technologies. It can add new sets of capabilities to already existing applications and even become part of the operating system or application environment. It is not in widespread use, nor has it been widely accepted as an inevitable trend. On the one hand, the current state of agent technology shows that agent technology is still an active

research area, as emerging isolated pioneer products show, i.e. the full set of technologies are not yet available, nor do they integrate with one another. On the other hand, prototypes and early adapters demonstrate the value of agent technology. The deployment of agent-based systems and agent technology increases and commercial tool sets for building agents enter the market.

The state of the art of commercial and non commercial mobile agent systems is represented by Java based systems, such as Concordia (Mitsubishi, [Con97]), Aglets Workbench (IBM, [LaOs98]), Agent Development Kit (Tryllian, [ADK02]), Grasshopper (IKV++, [IKV03]), or JADE (an Open Source project from TILAB, [BCPR03]). These agent systems vary widely in functionality. Although the latter two adhere to the FIPA standard, agents built for the one system do not execute on the other, nor do the agents communicate across the different systems. Table 2-15 is taken from [OMG00] and gives an overview of commercial agent systems and environments.

With this brief overview of the standardization bodies' effort and of the wide variety of commercial and scientific agent systems, the focus of the architecture becomes a fundamental aspect for enclosure and differentiation. Since the start of the development of the Java Agent Environment in early 1996, which has been accompanied by the standardisation processes, the focus was on distributed and autonomous trading facilities of mobile agents and the strong alignment towards mobile computing. Therefore, the next two chapters describe in detail the architecture, additional agent system components, and services such as maintenance services for mobile agents and the agent directory which are necessary to ensure autonomous work of mobile agents.

Chapter 3

The Java Agent Environment - JAE

The Java Agent Environment - JAE is an infrastructure supporting mobile agent technology that has been developed at the *Rheinisch-Westfälische Technische Hochschule Aachen* (RWTH Aachen), department of Computer Science, since early 1996 [PKL97]. The main focus of JAE is on the so-called *service-based agent programming* paradigm and the support of service trading by dynamically looking up services. Look up functions have been clearly identified by both relevant standards, the OMG MASIF and FIPA, as necessary to support agents in mobile agent systems. Agents working autonomously in an open agent infrastructure need to find other agents and services. Particularly, inter-operable services for mobile agents in an open environment demand more than just a simple look up function. Both standards do not recommend an explicit service trader or directory architecture. Therefore, the early research work on distributed and autonomous service trading for mobile agents [PaLe97] and the experience with the proprietary architecture and the prototype implementation of the trading facilities will be discussed in the following sections. Supplementary, the architecture of the agent system has been designed to support a variety of mobile devices designed for different requirements, including a simple mobile phone, a *Personal Digital Assistant* (PDA) or a high-end multimedia notebook with wireless access to the network. To sufficiently support unreliable, narrowband, wireless access networks as well as high speed network connections a lot of effort has been put into the design and the implementation of the *Agent Transfer Protocol* (ATP). The combination of the service trading facilities and the ATP has lead to the concept of *plug-and-participate* of agent systems in JAE.

Chapter three starts with an overview of the JAE architecture and its framework for developing and deploying mobile agents. Subsequently, the core technology for agent mobility, agent security, agent services and agent communication is described. Other focuses of this chapter are the service trading facilities introduced by the so-called *service center* with implementation details, measurements, and programming examples to mediate the service-based agent programming paradigm as well as showing the easy implementation of user created agents.

3.1 Overview of the Architecture

An infrastructure supporting mobile agent technology requires the distribution of the agent-based middleware to all participating host machines. Currently, most mobile agent systems are based on Java, thus supporting heterogeneous host systems, mobile objects, and network communication. Solutions based on Java become scalable and data also remain useable without difficulties in adapting. The use of Java as the implementation language for the agent system enables easy access to the Internet, thus making services provided through the Internet easily accessible to any agent-based applications. Obviously, the *Java Virtual Machine* (JVM) is the most important foundation for the agent-based middleware. The decision to use Java as the agent language and to implement JAE upon a JVM was simple, considering that the JVM is not necessarily software, but can be a processor in a hand-held device in the future. Recalling the general coarse architecture described in figure 2-7, the following figure 3-1 shows the adapted overview of the proposed agent middleware based on Java.

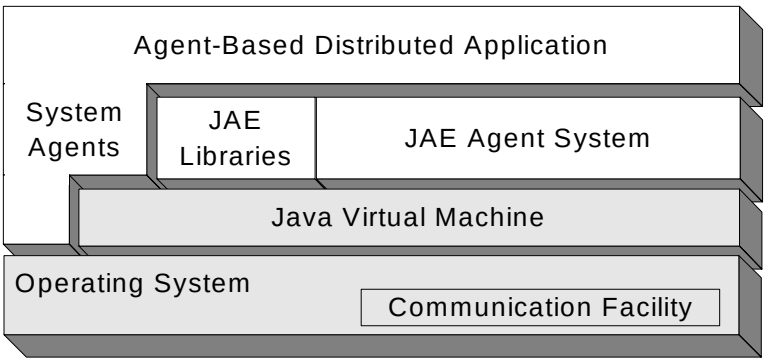

Figure 3-1: Coarse Architecture of the Java-based Agent Middleware

The agent system is implemented on top of the JVM, the operating system, and the communication facility. It is running permanently to handle and host the mobile and system agents. The JAE class libraries contain e.g. additional agent system classes, and communication facility classes, which serve both the agent system and the agent-based applications. Typically, an agent-based distributed application will be a combination of mobile and system agents, Java applications, and platform dependent programs.

One essential decision that comes with the remote programming paradigm that allows service-based agent programming is the strict differentiation between mobile and system agents. This concept of open market services and remote programming through mobile agents is not really new to the field of distributed computing. The general idea of an open market of services is that the application is separated from the resources needed to fulfil its task. These resources are modelled by services, which are independent of the application. Thus, the service can be used also by other applications [TWH90]. This leads to a scenario where competing service providers offer their services to a large group of agent based applications. Together with the remote

programming concept, this results in mobile agents roaming through the agent network using the services they need to accomplish their tasks. Agents supporting applications and agents offering services share certain attributes, but differ in certain others. In JAE system agents usually offer services to mobile agents and they are implemented completely in Java or simply act as a wrapper of already existing legacy applications or services. Within JAE, mobile agents are not allowed to directly access local resources, the network, or even remote hosts or services. The services a mobile agent can rely on while roaming through the agent network thus become fundamental. It is very important for mobile agents to find services that help to complete their tasks. A common way to determine the destination of mobile agents in other realizations of agent systems is an explicit addressing scheme, which of course is not as flexible as destination determination at run time of mobile agents. Before going into the details of the trading facilities, a closer look at the modules of the JAE agent system is necessary.

3.1.1 Agent System

The interconnection of agent systems in a network establishes the basis for distributed applications. The agent system is started as a resident process on each host that should provide access for mobile agents and provides the execution facility for all agents. The core components of the JAE agent system are depicted in figure 3-2. The mandatory components *agent manager*, *service center*, and *communication manager* can be also found in the reference model of FIPA with different names. The additional components *security manager*, *system administration*, and *persistence storage* are important and necessary implementations of the agent system,

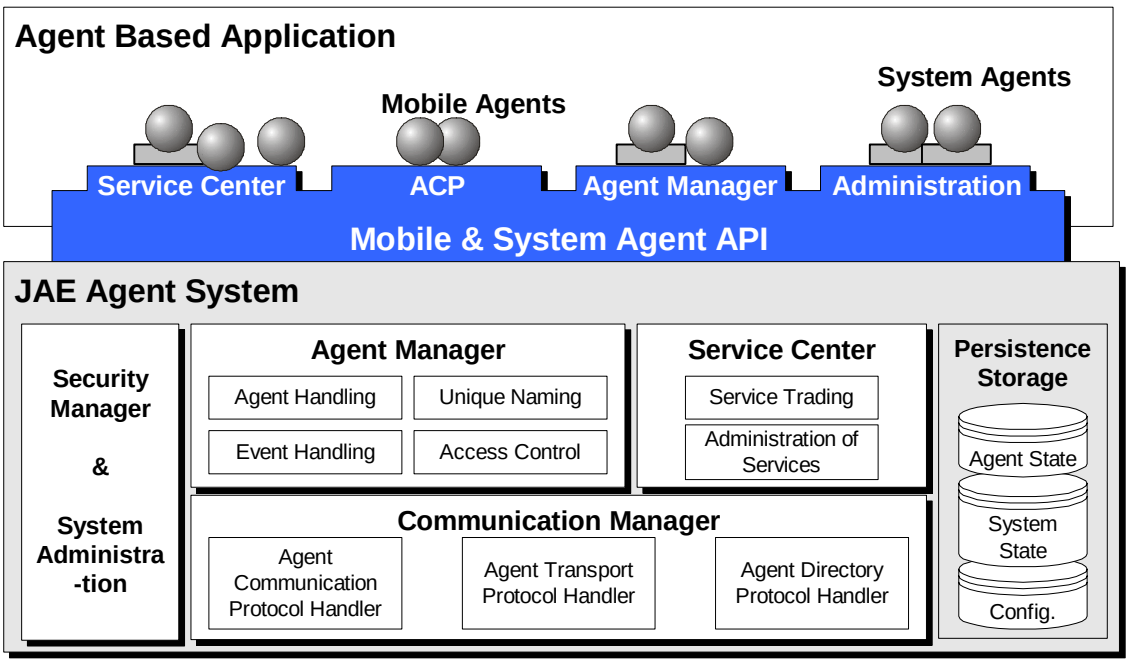

Figure 3-2: The Architecture of the JAE Agent System

which emphasize the explicit realization of this agent system in comparison to a reference model. The mobile and system agent *Application Programming Interface* (API) is defined by these components and allows the access to specific agent system components. Depending on the agent type, access to some specific API is only granted to system agents, e.g. the administration API. A subset of the available system interfaces to the service center - the *Agent Communication Protocol* (ACP) and the agent manager - is offered to both the mobile and system agents. The agent system interfaces the communication protocols through its communication manager and its service center, which in turn connects to the available networks and their bearer services. The protocol handlers of the communication manager establish the protocols used for inter-agent communication, for agent migration, and for accessing the *agent directory.*

System Administration

The agent system itself has to be configured and managed. This is done by the system administration module that reads and stores system relevant information from a configuration file. Agent system relevant information such as locally available services, path structure, default values, and agents to be automatically started with the system boot up are stored in the configuration file. The additional *Graphical User Interface* (GUI) allows the administration of the agent system during execution, which includes the management of mobile and system agents. A stand alone monitoring tool further visualizes all agent systems in the network and shows the activities of mobile and system agents.

Agent Manager

The agent manager keeps track of the agent's life cycle, which starts with the configuration and creation and terminates after fulfilment of the agent's work somewhere in the agent network. Each new agent is started with initial arguments and must have a unique identification in the whole agent network. Furthermore, each re-start of a mobile agent needs an access control and on termination, its results secured appropriately. As prior described in figure 2-8, the life cycle of an agent has different states, such as suspension, execution, migration, or service look up, which have to be managed by the agent manager with the help of the event and agent handler. In close relation to the security manager, all mobile codes are signed and the number of executed agents are checked with the maximum number of allowed agents on the agent system. The limit of agents per system can be changed dynamically by the system administration GUI due to the fact that the resource consumption of agents cannot be predicted. The facilities to support mobile devices by the opportunity of remote invocation of agents are implemented in the agent manager as well. The idea of the remote invocation mechanisms is described in chapter 4.3.3.

Service Center

The service center takes care of the administration of local services and provides a trading mechanism for remote services. Services are always looked up dynamically and invoked

through the use of the service center. A mobile agent gets in contact with a specific service agent by submitting requests to the service center with descriptions of required services. The service center either mediates the requests to a suitable local system agent or provides information about another agent system that offers the requested service. Furthermore, it registers and de-registers agent systems as well as mobile and system agents with a central repository, the agent directory. This agent directory is the central service reference and naming repository for all available services.

Communication Manager

The communication manager encapsulates all relevant application protocols for the agent system. Various inter-agent communications are combined into the *Agent Communication Protocol* (ACP), which uses e.g. asynchronous one-way agent-to-agent messages, synchronous two-way agent-to-agent messages as well as the blackboard communication. The most relevant communication concepts are the message passing and the blackboard communication, both of which have been implemented in JAE. An important status preserves the *Agent Transport Protocol* (ATP), which is the key for agent migration from one physical host to another and therefore must consider security aspects in particular. Finally, the *Agent Directory Protocol* (ADP) is introduced so that in close conjunction to the service center, the access to the agent directory is granted. The overall structure of the agent environment and the protocols is depicted in figure 3-3.

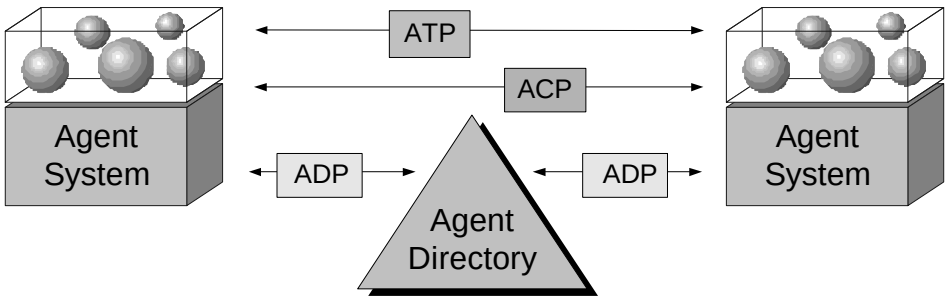

Figure 3-3: JAE Components and Protocols

Security Manager & Persistence Storage

Both the security manager and the persistence storage affect the whole agent system and therefore are not considered as separate components. The persistence storage promotes the security manager as well as the communication and agent manager. It supports agents and systems with mechanisms for persistence storage in case of physical damages to the host or physical interruption of the interconnections. Mobile agents introducing mobile code are especially subject to strong security restrictions which are enforced by the security manager. Interaction with other agents or the usage of system resources can only be achieved through the agent manager and the access control, which are inheritances of the security classes of the *Java Development*

Kit Version 1.2 (JDK, [JDK12]).The security manager also handles the management of private and public keys as well as certificates. Due to the trust model of JAE, which is in close context to the keys and certificates, access to them occur during run time. Finally, the recovering steps of a failed system must be handled by the security system and will be discussed in detail in one of the following sections.

Figure 3-4 depicts a perspective of the main implementation modules of the JAE agent system. The hierarchical representation is subdivided into 3 levels showing the relations of the core modules to the infrastructure and implementation modules. For the sake of simplicity, the conjunctions between the modules in each level have been omitted.

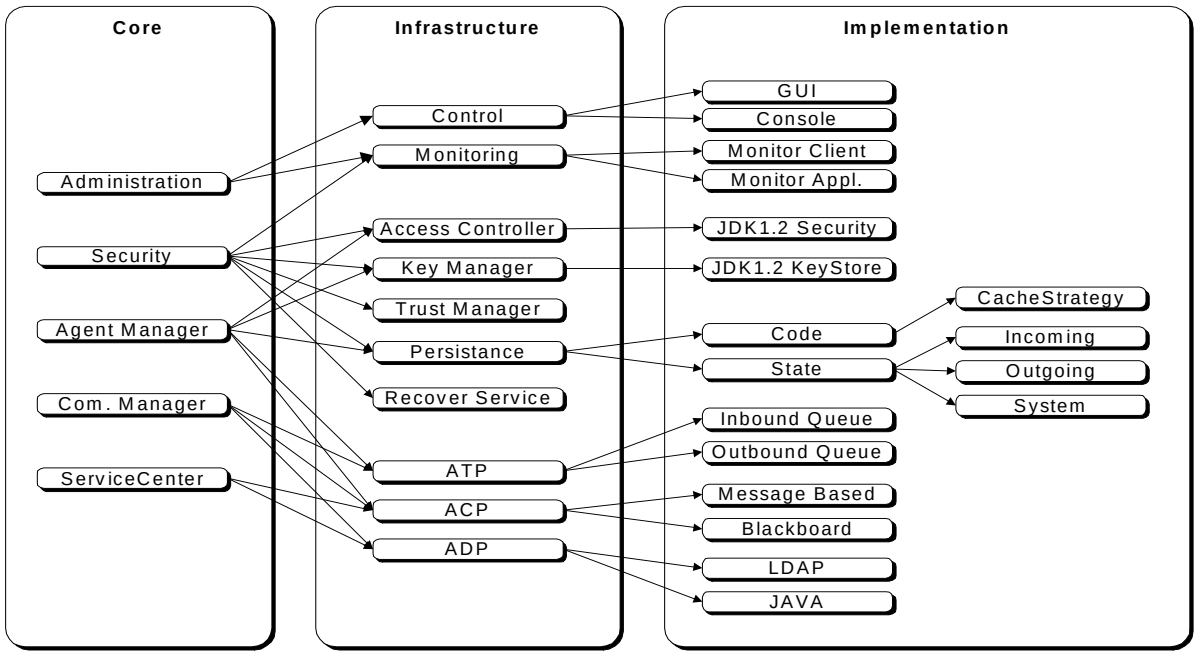

Figure 3-4: Hierarchical Module Representation of the JAE Agent System Implementation

3.1.2 Agent Types

The distinction between mobile and system agents is reflected in the agent class structure of JAE. Both mobile and system agents are derived from one basis class and thus have the same core states: configuration/creation, active, waiting/suspend, unknown, and terminated. Depending on the mobile or system inheritance, additional states have been joined. Due to the unique ability of mobile agents to migrate to different agent systems, two types of migration mechanisms have been implemented: the sequential migration (*GO-method*) and the parallel migration (*SEND-method*). The sequential migration follows the general migration process steps (see figure 2-5), whereas the parallel migration combines two steps: cloning of the parent mobile agent and its parallel migration to remote hosts for simultaneous execution. The clones with the same type and state as the original return to the parent agent as soon as their task has been fulfilled and transfer each result to the waiting parent agent.

System agents do not have the possibility to migrate and therefore, their code is considered trustworthy, i.e. system agents do not underlie the tight security restrictions of mobile agents. These agents have full access to the resources of the hosts and must be executed by the administrator of the agent system. Service agents are a sub-type of system agents. They provide services for mobile or other service agents and are enriched by methods to access the service center and are automatically registered with the service center upon creation.

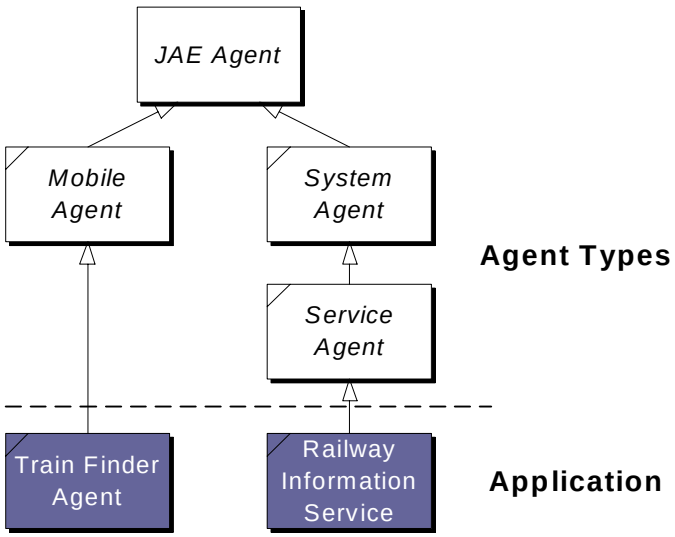

Figure 3-5: Example of a Mobile and System Agent Class Structure in JAE

Figure 3-5 shows the general class structure of the agent types in JAE. In this example, a distributed railway information application is considered. A mobile agent takes over the task to gather information about requested trains and searches for this train information, which is offered by several distributed service agents. The service can be e.g. a client for local database requests and can consist of simple classes, but a full Java class package with lots of internal classes is likely as well. In this case, the sub-type of the service agent provides the interface. This generic service agent concept allows to support any legacy applications very easily - the service agent is simply a Java wrapper to some application interfaces. In JAE, two further types of service agents are distinguished, i.e. passive and active services. Active service agents follow the general state model. After the invocation, an *initialize()* method is executed, followed by the *live()* method and a *die()* method. These active service agents are suitable for continuously executed services, which are not influenced by other agents, e.g. checking a mailbox account. Passive service agents have omitted the *live()* method, thus the states of these agents remain inactive until other agents request the offered methods.

For a better understanding, the life cycle of the JAE mobile agent is introduced in detail. At any given time, the execution flow of the program is in a certain state. Transitions between the states occur through different actions. The basic state transition at the beginning, which is common to all agents is the *initialize()* method. The state changes from the *configuration and crea-*

tion state to the *active* state. However, at start time the agents have to be configured. Depending on the application, the parameters can be requested with the help of the agent system GUI or must be separated from the agent. The reason is, some applications are not meant for direct user interaction. Therefore, the agent system provides an API that allows to hand over parameters to the other agents during run-time. With the *live()* method the mobile or the active service agent is executed, in which the agent developer has coded the desired behaviour of the agent. From this active state further seven states are operational for a mobile agent. These states and their transitions are depicted in figure 3-6.

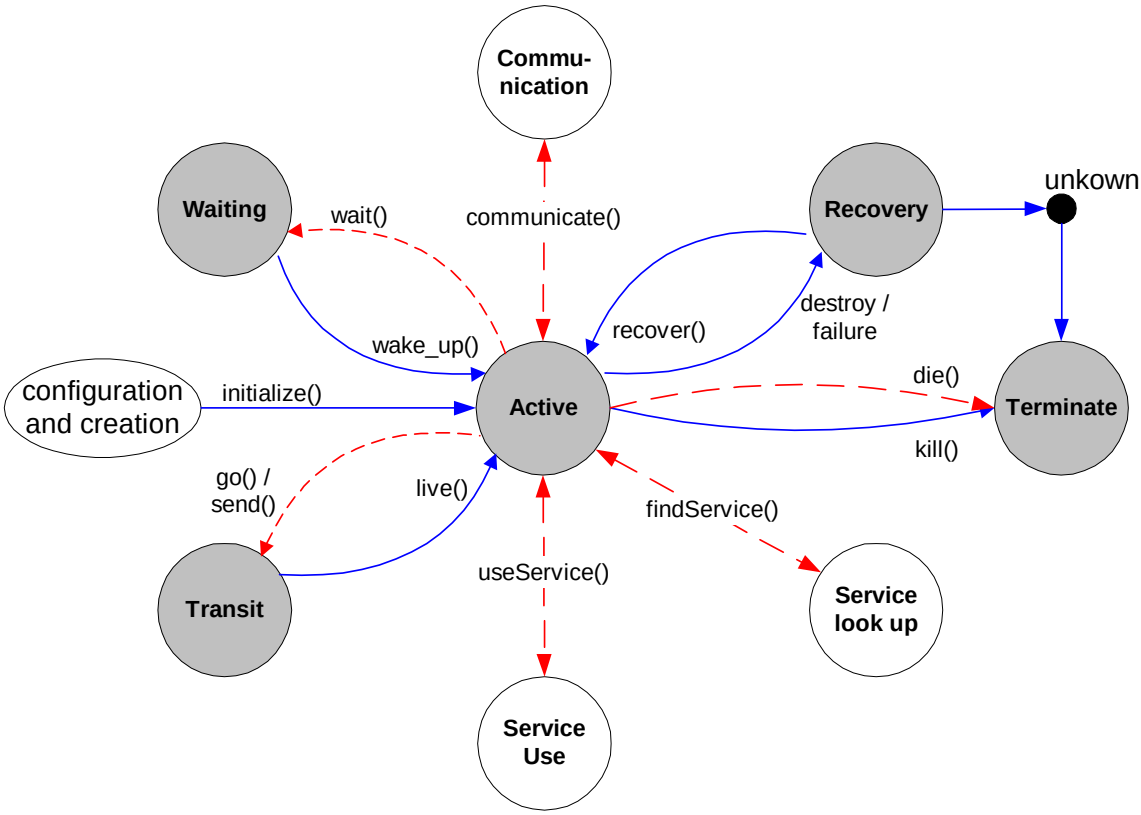

Figure 3-6: JAE Mobile Agent Life Cycle

In the diagram, agent system initiated states are shown as gray circles and their transition methods are differentiated by a solid arrow. Only one transition can be forced by the agent system, which is the *kill()* method for any shutdown or system relevant reasons. Then the agent turns to the state *terminate* just as with the agent initiated *die()* method. Another meaningful state is achieved if in any state an error or failure occurs. In this case, the agent system tries to recover the agent. When the recovery does not maintain, the state changes from *recovery* state to unknown and terminates. These execution states are controlled by the agent system to handle the agents correctly, according to their current state of execution. All other states are achieved by agent originated method calls. Two more states, which are also shown in the general agent life cycle, are the *transit* and *waiting* state. In both cases, the activation of the agent

has to be initiated by the agent system. The waiting agent has to be activated on some events (*wake_up()*) and the mobile agent has to be re-started (*live()*) on a new agent system. The remaining three states are achieved by the agents themselves through communication activities with other agents or the service based programming (*findService()/useService()*).

To resume a mobile agent after a migration, the agent should continue with the next instruction of its code after a *go()* or *send()* command. As a direct access to the program counter in Java is not possible, the weak migration is enabled with the introduction of a counter (*getHops()*). This counter is an object member variable that can be accessed by the Java Object Serialization API. After each migration, this variable is increased, enabling mobile agents, during runtime, to decide where to proceed with the program code in the *live()* method. To conclude this section, an example of a code segment is given. The mobile agent uses a translation service to translate a submitted text. During the creation state, the text can be achieved, e.g. by the GUI of the agent system.

```
01 public void live() throws java.io.IOException
02 {
03     switch(getHops()){
04     case 0:
05          service=getServiceCenter().findService(query);
06          go(service.getTicket());
07          break;
08     case 1:
09          result=(String) getServiceCenter().useService(service);
10          setResult("TRANSLATION",result);
11          goHome();
12          break;
13     }
14 }
```

Program 3-7: Code Fragment of a Mobile Agent Example

The *live()* method of the mobile agent distinguishes two cases. The code in the first case is executed after the agent invocation. At this state, a request is initiated to the service center demanding a service agent that offers a translation service. The result contains the description (*getTicket()*) to the agent system hosting such a service. After the *go()* method, the mobile agent might have migrated to another system (not necessarily if the service is available locally) and the second instruction block is executed. The mobile agent invokes the received method of the translation service agent via the service center and stores the results (line 10). Finally, it returns to the initial system and waits for the results to be requested by a user or an application.

3.1.3 Agent Monitor

The last sections have introduced JAE as a distributed system with system and mobile agents running on different hosts. There is an obvious need for a developing tool that can visualize the JAE network. The JAE monitor shows agent systems running on different hosts and agents on these systems as well as currently moving agents. The monitor is an important development tool of the JAE framework, but on the other hand can be used for demonstration purposes. It provides developers with a central console to view messages collected from travelling agents and the agent system, and allows them to follow agents on their way through the agent net-

work. There is no restriction to the number of active monitors in the JAE network. The administrator of each agent system keeps the information (there can be none) to which monitor the agent system should report its activities. The monitor client has been implemented as a system agent and can be dynamically started if required. The information about the agent system and agents as they are created, moved or destroyed is reported by the monitor client using the *Remote Method Invocation* (RMI, [RMI03]) of Java. The reasons for RMI are simplicity, extensibility and compliance with a standard. RMI allows calling a method of a remote object and receive its results. Furthermore, new functions to the monitor can be introduced with only little effort.

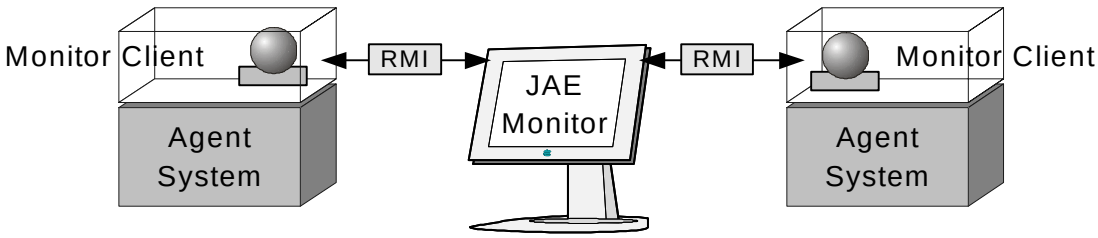

Figure 3-8: Overview of the Monitor Components

The monitor package of the JAE library contains both the monitor application with the modules needed for the graphical user interface and the monitor client - an inheritance of the system agent. The system agent completely encapsulates the RMI protocol to the monitor, allowing additional or other protocols (e.g. due to performance problems). The monitor displays a near real-time view of the JAE activities. All movements of mobile agents are displayed on the monitor, which means that an icon representing the agent in action appears on the GUI of the agent system. The disadvantage of the RMI protocol is that the caller is blocked until the remote call on the monitor has been completed. This synchronous protocol may become a problem in wide area networks with a slow data connection and therefore must be enhanced with an asynchronous protocol, e.g. the *Simple Network Management Protocol* (SNMP) to remain on an Internet standard and have the opportunity to match existing monitoring tools.

3.2 Service Trading in JAE

The basic idea of JAE's service trading has been taken from the *Open Distributed Processing* (ODP) reference model. The so-called ODP trader has been defined in [X.950]. The essential difference to the classical service trading in client/server systems as shown in figure 3-9 is that the role as importer of services or exporter of services is not fixed to one object. The trader itself can act as an importer and exporter of services and, in conjunction to other traders, build a trading community. The key to a successful service trading is the service description of a service exporter. Standardized services are exported by a so-called service offer, which is

unambiguously defined by its service type, its interface type, its service properties, and its service offer properties. Important to service trading in mobile agent systems is that many of the research work of the ODP trading can be transferred to the service-based agent programming; more details to ODP trading can also be taken from [SPM94]. The OPD trading model can be found with some adaptation in JAE.

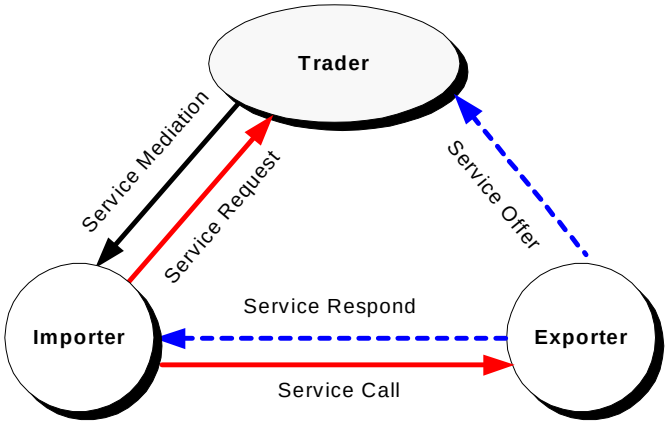

Figure 3-9: Classical Client/Server Service Trading

The support of autonomous migration of mobile agents in an agent system network demands for a component like the trader in ODP. Thus, mobile agents with a service description have the possibility to find a service, move to its location and execute the required service during run-time. Service addressing in existing agent systems has different approaches, some of them are listed following: the agent system *Mole*, for example, has a very simple service access, i.e. services are directly implemented in the mobile agent. A similar direct target addressing of services is implemented with the *Aglets* agent system; the operations a mobile agent should fulfil are coded in a script [LaOs98]. A theoretical modification of the target addressing scheme should allow enhancing the number of alternative service locations in the script during run-time so that, depending on achieved results, a new location can be calculated by the agent. More theoretical approaches are listed in the publication of [LiDr95], for example, the idea of the *functional neighbourhood* is that agents are only aware of other agents with similar or extending services. The whole network of agents let mobile agents dynamically look for their functions to solve their task. Another concept is the *semantic routing*, which requires the analysis of the agent's task to decide with the help of a service list carried with the agent, which services to access. To analyse the task of an agent, it is proposed to use meta keywords or a service description similar to the ODP service description without a concrete implementation. In JAE the neglected, but very important subject of service trading has been taken up and a practicable solution has been implemented with the introduction of the service center.

3.2.1 The Service Center

The main focus of the service center lies upon service trading and the easy access to various services for mobile agents. The service center is in charge of the administration of local services and provides a trading mechanism for remote services. The mediator for mobile and system agents is an important constituent of each agent system within JAE. The service center helps to simplify or even eliminate configuration needs for newly introduced services. It helps to decentralize the responsibility for administering the agent services and introduces fault tolerance simultaneously, e.g. suppose it is desired to change a telecommunication service of an agent system. Given current practice, one would inform all domain members by email about the unavailability of a service, and then they would have to wait until the update of the service. With the service center concept, non of this inconvenience would be necessary because if the telecommunication service is not available, another one might advertise an equivalent service and this other available service will be used automatically. Following main requirements to JAE have been defined:

- mobile agents must be able to navigate through the agent network
- mobile agents must be able to find services and other agents they are communicating with
- agent systems must be able to sense and react to the network environment, so that mobile agents may act autonomously without further user interaction.

Since the agent developer or agent user is not aware of where their agents can fulfil some tasks, there must be an instance that offers this information. The service center introduces the idea of distributed trading. Instead of one central trader instance, each agent system has its own trading mechanism which has knowledge of the whole JAE environment. There are mainly three different functions fulfilled by the service center:

- firstly, it provides agent administration, i.e. registering and de-registering of agents running on the same agent system both to the local database and the agent directory.
- Secondly, it acts as a trader, providing services to mobile and system agents.
- Last but not least, remote service calls should be handled by the service center as well, but are not implemented yet.

To store all required data, the service center has access to the agent directory that contains data about all available services and mobile agents in the whole agent network. The agent directory is the central repository which can be a common database, the CORBA naming service, or an X.500 directory service [X.500]. In the JAE implementation, two versions of an agent directory are used. On the one hand, the *Standalone LDAP Access Protocol Daemon* (SLAPD, [SLA96]) is used as agent directory to have a lightweight and open available quasi standardized directory service instead of the X.500. The *Lightweight Directory Access Protocol*

(LDAP, [RFC2251]) itself is much easier to handle and wide spread in use. On the other hand, for measurement purposes, a Java-based agent directory has been implemented. All components communicate through well-defined interfaces, and therefore, the integration of new trading mechanisms or directory services is fairly easy and does not affect other components of the service center.

The service center has a common API to all agents and provides functions necessary to access the agent directory suitable for both mobile and system agents. The service center API is completely separated from the agent directory. Although system agents have access to local resources and the operating system, mobile agents are only allowed to access services through the service center API. Consequently, the service center plays a central role and provides services, agent administration as well as agent system administration, i.e. registering and de-registering of agents and agent systems. Certainly, it also acts as a trader and communicates with different implementations of agent directories (e.g. the Java-based agent directory and the SLAPD).

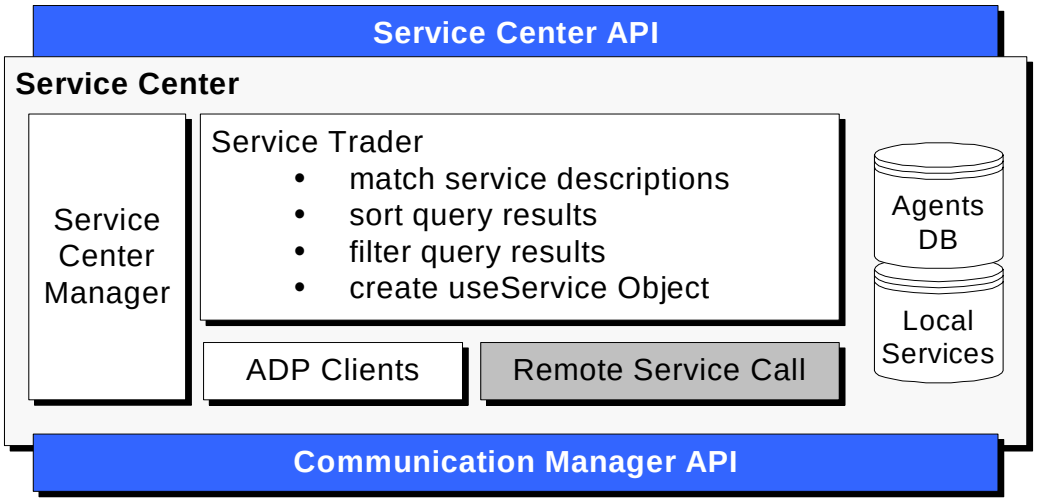

Figure 3-10: Architecture of the Service Center

Figure 3-10 shows the service center architecture with its main components. The dotted gray box marks the not yet implemented remote service call component. The service center API provides all functions of the service center to mobile and system agents and is independent of the implemented trading mechanism. The service trader is the mediator between the agent's requests and service descriptions of service agents stored in the agent directory. The trader performs four functional steps to prepare the results of the database search. It generates the *useService* object that contains all information about the location of the service and the requested service agent methods. These steps are in detail:

- **match service descriptions**: the results of a successful search in the agent directory or the local database are *QueryResult* objects. In a first step, the description parameters of these objects are matched against the request of the mobile agent.
- **sort query results:** in the second step, the ranked *QueryResult* objects are sorted according to their relevance.
- **filter query results:** *QueryResult* objects without any match of the service description parameters are deleted.
- **create *UseService* Object:** in the last step, the *UseService* object is generated in the order of the remaining *QueryResult* objects. This object is used by the caller agent through the *useService()* method.

The dynamic look up of service types is supported by hierarchically sorted keywords that allow different levels of search strategies. To have a faster look up mechanism, local service descriptions are also stored locally in each agent system. This prevents the time expensive look up in the agent directory, because the service center first searches in the local service database and, only if necessary, requests the agent directory via the ADP client. The same holds for look up of agents in the JAE network. The local agent reference database is used by the agent manager as well as for agent communication aspects.

Each ADP client forms a separate module, thus new protocols or agent directory services can be adapted easily. A new ADP client has to implement the five main operations of the abstract *ADPClient* class:

- **read:** to obtain pre-selected *QueryResult* objects from the local database or the agent directory that contain service description parameters as hierarchically sorted keywords. Descriptions of the agent system and mobile agents must be accessible as well.
- **write:** to register service agents by means of service description objects, the agent system, and mobile agents locally and remotely with the agent directory.
- **delete:** to de-register service and mobile agents as well as the agent system from the database.
- **check:** to test the availability of agent systems, which are not accessible due to host or network crash.
- **add domain:** to extend the agent directory by an additional domain

The ADP client exchanges information via the communication manager interface that encapsulates the implementation of the agent directory protocols. In case the agent directory is the SLAPD service directory, the ADP is equal to the LDAP. For the Java-based service directory, a proprietary object application protocol has been implemented.

Summarizing, the service center manager takes care of the above mentioned administration tasks, which can be enumerated in six main functions:

- **Service Administration**: registration and de-registration of services.
- **Mobile Agent Administration**: registration and de-registration of migrating agents
- **Agent System Administration**: registration and de-registration of agent systems and testing of availability.
- **Service look up**: look up and mediation of services and mobile agents
- **Service Execution**: execution of the requested service agent.
- **Remote Service Call:** support of transparent remote service call

3.2.2 Service Implementation Example

Services are described with the *service description* objects. These objects enable mobile agents to find and use them by means of the service center. A service description contains all information about a service agent. The classified service description that is proposed in JAE is similar to ODP trading specification. The service center uses attribute-value pairs to store all this information. There are 4 attributes specifying a service.

- **Service Type**: Description of the service category.
- **Action**: Functions offered by this service.
- **Parameters**: Parameters accepted or needed by the service.
- **Return Values**: Type values returned by the service functions.

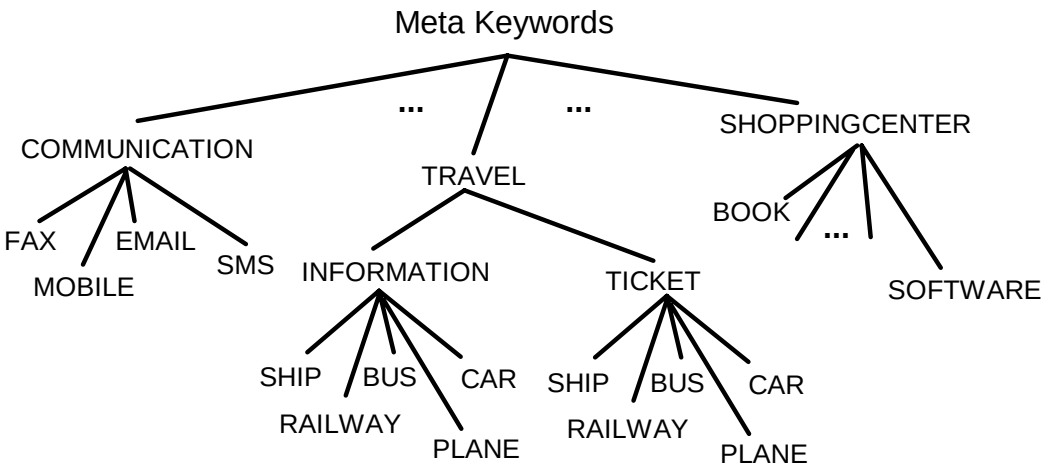

Figure 3-11: Example of Hierarchically ordered Meta Keywords describing Service Types

To describe these attributes a common base of keywords have to be established - the so-called *meta keywords*. These keywords allow the agent developers to specify their services. Meta keywords are global to one domain or even the whole JAE network describing all available services in a hierarchical structure. An example of hierarchically ordered meta keywords describing some services are shown in figure 3-11.

Service agents use these keywords to describe the services they offer, while mobile agents need them for specifying the service they want to use. For each of these attributes, a separate set of meta keywords exists. They are pre-defined, but can be extended dynamically, capable to describe all available service types, actions, parameters and return values to be used in the domain or the environment. If a service agent developer needs additional meta keywords to describe a new service, the list of keywords can be extended and have to be made public to all agent programmers. Values (i.e. meta keywords) for the service type attribute are provided hierarchically, e.g.:

- TRAVEL.INFORMATION.PLANE
- TRAVEL.INFORMATION.RAILWAY
- TRAVEL.GUIDE.EUROPE
- TRAVEL.GUIDE.ASIA

They generally describe the service category and enable different levels of requests, e.g. a mobile agent can request a service with only the coarse type description “TRAVEL”, receiving many results to service agents in comparison to a more detailed request: TRAVEL.GUIDE.EUROPE. The following example will enlighten the idea of the meta keywords.

The service type of the example *Translate* is LANGUAGE.TRANSLATION. Each service can provide several different functions, so that they have to be described with a second attribute called action. For each service function provided, one or more action values must be set. The action attributes for the *Translate* service are:

- `TRANSLATE` (translating a message)
- `SPELL` (spell a word in a language)

The attributes *service type* and *action* describe the functions offered by a service agent. Now, additional information is needed to specify the service properties. This information is necessary to specify the method call of services, i.e. which parameters are needed or accepted by a specific service and what kind of results will be returned from each service function. Therefore, a list of parameters are necessary in the *service description* object, again using the *meta keywords*. For the *Translate* service example, the parameter attributes are:

- `TO` (the language to translate to)
- `FROM` (the language to translate from)
- `MESSAGE` (the message to be translated or spelled)

Finally, the return types for the two actions TRANSLATE and SPELL are:

- `STRING` (contains the translated text)

- `BYTE[]` (Byte array, contains the spelled word as an audio file)

In addition to the description of the service, the name of the service class and the name of the methods for each action have to be included. The complete *service description* object of the *Translate* service agent is listed in Program 3-12.

```
01 s=new ServiceDescription("Translate",         // class name
02                          "LANGUAGE.TRANSLATION", // type
03                          "transText,spellText",  // methods
04                          "TRANSLATE,SPELL",      // actions
05                          "FROM,TO,MESSAGE",      // parameters
06                          "STRING,BYTE[]");       // return values
```

Program 3-12: Example of a Service Description in JAE

Having introduced the *meta keywords* and the *service description*, a closer look to the administration of service agents is required. When initially started, new service agents are automatically registered at the service center using the *registerService()* method. Therefore, their *service description* has to be extended with additional information about the location of the hosting agent system and a unique identifier for the service agent. The *service description* is now stored in a list of locally available services, together with a reference to the service agent itself. Additionally, the service description is stored in the agent directory, exporting the new service agent to all agent systems in the JAE network. Subsequently, the new service can be used from each agent within the environment. If a service is not available any more, the *deregisterService()* method removes the service from the agent directory and avoids that agents migrate to non-existing resources. Note that registration and de-registration must be invoked automatically by the agent system during start-up and shutdown of service agents. The service registration process is depicted in figure 3-13.

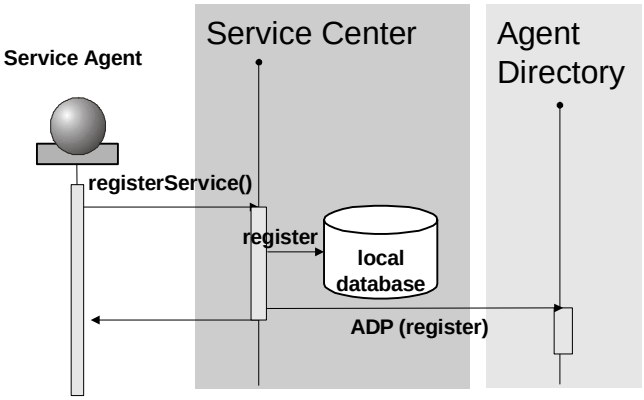

Figure 3-13: Service Registration Process

Having demonstrated how service agents are registered to the service center, the next step is to show how a mobile agent can find services. The location of the service agent remains transparent to the mobile agents. When a new mobile agent is started, only the type of the required service is known, but not its location. Therefore, the mobile agent has to request the service center to receive a *useService()* object to other agent systems hosting the requested service

agents. The query sent to the service center contains the requested *service type,* the *action* and the accepted *return type,* given as *meta keywords*. Additionally, the query includes a property list with all parameters and their required values by the service agent. The parameters are stored as key-value pairs. The key is a *meta keyword,* whereas the value is a data string. In addition, the service center provides the *ControlBlock* object to allow variable look ups in the agent directory. Since it is possible that an agent needs more than one service in order to fulfil its operation, it can use the *ControlBlock* Object to specify the number of services requested. If more than one service is inquired, the *findServices(ControlBlock)* method returns an array of *UseService* objects. The agent programmer has to decide himself how many of these offered services the mobile agent should request. The mobile agent uses the *findServices()* method provided by the service center to look for specific services that match to the *meta keywords* describing the desired service. This is performed in three steps (see also figure 3-14). The service center first looks up the list of local services to find out if a service agent is locally available. If no such service is available locally, in a second step a request is sent to the agent directory to receive information about all registered services of the whole JAE network matching the keywords.

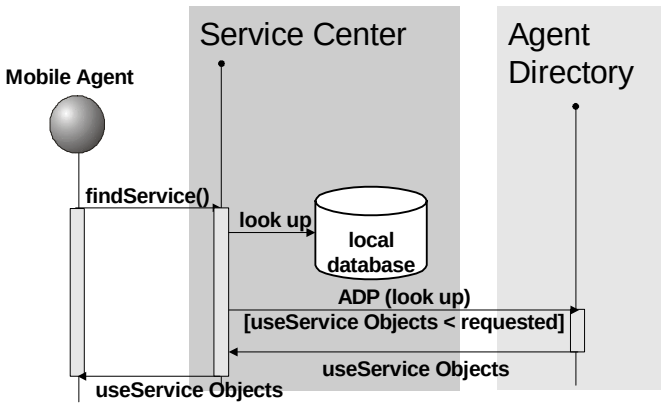

Figure 3-14: Service look up Process

The final step includes some additional selection criteria. So far, it has been clarified that the received services provide the requested functions, but it is not clear if the services can cope with the actual parameters of the mobile agent. Therefore, those services have to be matched against the parameters submitted by the mobile agent. The service center selects services that match best to the mobile agents parameter list (i.e. the service which has the highest number of matching parameters) and generates a *UseService* object. This also contains information that is required for the agent to use the *go()* method, and additional information for the other service centers to invoke suitable service agents with the actual parameters of the mobile agent. Thus, after receiving one or more *UseService* objects, the mobile agent can migrate to the destination agent system where the service is located and uses the *useService()* method to access the service. During the initialization of a user interaction, the required parameters are requested, for

example *FROM="english", TO="german", MESSAGE="Merry Christmas"*, to get a translation of the phrase "Merry Christmas" into German. This information is stored in a property list (*parameter*), which is included in the query to the service center:

```
query=new Query("LANGUAGE.TRANSLATION","TRANSLATE","STRING",parameters);
```

In this case, "LANGUAGE.TRANSLATION" is the service type, "TRANSLATE" the requested action and "STRING" the accepted return type. The actual parameter values are stored in the property list *parameter.* After the mobile agent has reached the destination system where the requested service agent is located, it uses the *useService()* method to access the service agent. The mobile agent uses the *UseService* object received during the *findServives()* operation. The service center looks up the requested service in its list of local services and invokes the requested method through the Java Reflection API. Since service agents are only used in this way, it is ensured that no other methods can be used except those provided in the *service description*. In case the service center does not locally find the requested service and the specified location is a remote agent system, the *useService()* should initiate the remote service call. In other words, not explicitly migrating to the new location should indicate the service center to execute the remote service call. The latest version of JAE has not yet implemented this automatic switch to remote method invocation.

```
01 import jae.core.*;
02 import java.util.*
03 public class ServiceUser extends Agent implements MobileAgent
04 {
05  private UseService theService = null;
06  public void init(Vector parameters)
07  {
08   // any initialization
09  }
10  public void start()
11  {
12    // [...]
13    {
14      Properties props = new Properties();
15      props.put("MESSAGE", "The Sentence to translate.");
16
17      Query query = new Query("LANGUAGE.TRANSLATION",
18                              "TRANSLATE",
19                              "STRING", props);
20      theService = getServiceCenter().findService(query);
21      Ticket ticket = theService.getTicket();
22      if (ticket!=null) go(ticket);
23      else doSomethingElse();
24    }
25    // [...]
26    {
27      // do anything at the new location, e.g. call the service
28      Object result = getServiceCenter().useService(theService);
29      // [...]
30    }
31    // [...]
32  }
33  // [...] all other methods
34 } // end class
```

Program 3-15: Code Fragment for Service Access

For a better understanding of the service-based programming paradigm, an example of a service access and a service offer is described. The code example shows the usage of services through the service center and the requests for a service described with meta keywords. The

basic procedure is to build a query (Program 3-15, line 17) that describes the requested service and delivers the parameters through the property object (lines 14/15). The query is used by the service center to find an agent system that offers the requested service. The developer does not care about location and underlying network communication. The following *go()* method (line 22) is necessary to ensure the local service access of the agent which is the original intention of the mobile agent paradigm. The simple *useService()* call (line 28) executes the requested service at the new location that can be anywhere (local or remote). The Query describes the service type (“LANGUAGE.TRANSLATION”), the action that should be performed (“TRANSLATE”) and the return and input parameters (“STRING”, props). The service center searches for the requested service and returns a reference (line 20) that can be used to generate a Ticket (line 21).

Alternatively to the *go()* and *useService()* combination, in a next release JAE mobile agents should also use the remote invocation method with *useService(theService)* without the migration to a new location (again the programmer does not need to know where the service execution takes place, locally or remotely). The idea is to use a remote method invocation by simply skipping the lines 21 - 27 in the above given example and just to call the *useService()* method. The service center has the task to take over the communication setup and keep service calls transparent to the programmer.

```
[...]
20      theService = getServiceCenter().findService(query);
21      Object result = getServiceCenter().useService(theService);
[...]
```

Program 3-16: Code Example for Remote Method Invocation with the Service Center

Having introduced the mobile agent, it is still an open issue how service agents have to be implemented. Following, the counterpart of the mobile agent is listed. Any public method encapsulating the parameters in a property object can offer a method as an action of the service agent (see Program 3-17, line 10). The property object contains key-value pairs and is the

```
01 import jae.core.*;
02 import java.util.*;
03 public class Translator extends Agent implements SystemAgent
04 {
05   // [...]
06   public ServiceDescription getServiceDescription()
07   {
08     return new ServiceDescription("Translator",              // class name
                                     "LANGUAGE.TRANSLATION", // service type
                                     "translate, spell",     // methods
                                     "TRANSLATE, SPELL",     // offered actions
                                     "MESSAGE",              // parameters
                                     "STRING, BYTE[]");      // return values
09   }
10   public String translate(Properties para)
11   {
12          String translation = (String) para.getProperty("MESSAGE")// [...]
13          return translation;
14 }
15 //[..]
16 } // end class
```

Program 3-17: Code Fragment for Service Offer

interface to offer services. Keys in the property list are always the name of the parameter given as a meta keyword. The corresponding value is the actual parameter of the service method call. The service agent developer accesses the parameters with the standard property object methods. The required parameters can be obtained with the meta keywords (e.g. “MESSAGE”, lines 8 and 12), any other given parameters are ignored.

The agent service developer has to take care of any error handling routines if some essential parameters are missing or some type of inconsistency occurs. The implementation of the service agent is completed with the creation of the service description that is needed to register the service at the service center. The exportation of the actions and the service agent is transparently done through the service center. The final mandatory step is the implementation of the *getServiceDescription()* (line 6-9):

- **Service Name**: Class name.
- **Service Type**: Hierarchically ordered meta keywords from the list of meta keys used in a domain or an application specifying the type of the service. If no proper meta keyword describing the service exists, a new meta keyword can be introduced and must be public available within the domain.
- **Offered Methods**: All offered method names separated with commas within a string.
- **Service Actions**: Each offered method must be described with a meta keyword in the same order as in the previous parameter. Each meta keyword is again separated by commas.
- **Service Parameters**: A list of all accepted parameters are given as meta keywords within a string, separated with commas. The order of the keywords are irrelevant.
- **Return Value**: Each offered method requires a correspondent return type that is listed in the last parameter separated with commas.

3.2.3 Agent Directory

The agent’s place of execution depends on the service offered by the agent directory, and is decided at run-time. Accordingly, distribution of tasks remains completely transparent to the application developer, but an efficient handling of mobile agents’ queries is required. Assuming that the open agent environment consists of agents offering inter-operable services, and of agent-based applications using these various services, the service time of an agent directory becomes extremely important. Each agent of a distributed application contacts the agent directory at least once in its lifetime and thus, in a world-wide agent infrastructure, scalability and fault tolerance play a significant role.

As described so far, JAE is characterized by dynamically changing information and frequent information look up. Repeatedly changing information are birth, death and migration of mobile agents and agent system start and termination. Involved with these events are frequent infor-

mation look ups. The migration of a mobile agent causes an information look up in the service or the agent system profile to obtain the location of the destination. On migration, changes to the service and mobile agent profile are necessary. Generally, before agent migration can be executed, an information look up in the agent directory is necessary. The update of information proceeds when an action is fulfilled or while an action is in process. Hence, to speed up processes in the agent system, fast information retrieval is necessary. This introduces the following design criteria for the agent directory structure:

- Fast information retrieval
- Modifications of agent system information and service description are not time critical because these modifications happen infrequently.
- Modification of the mobile agent profile is time critical since modifications occur quite often.
- Keep it simple and intuitively understandable

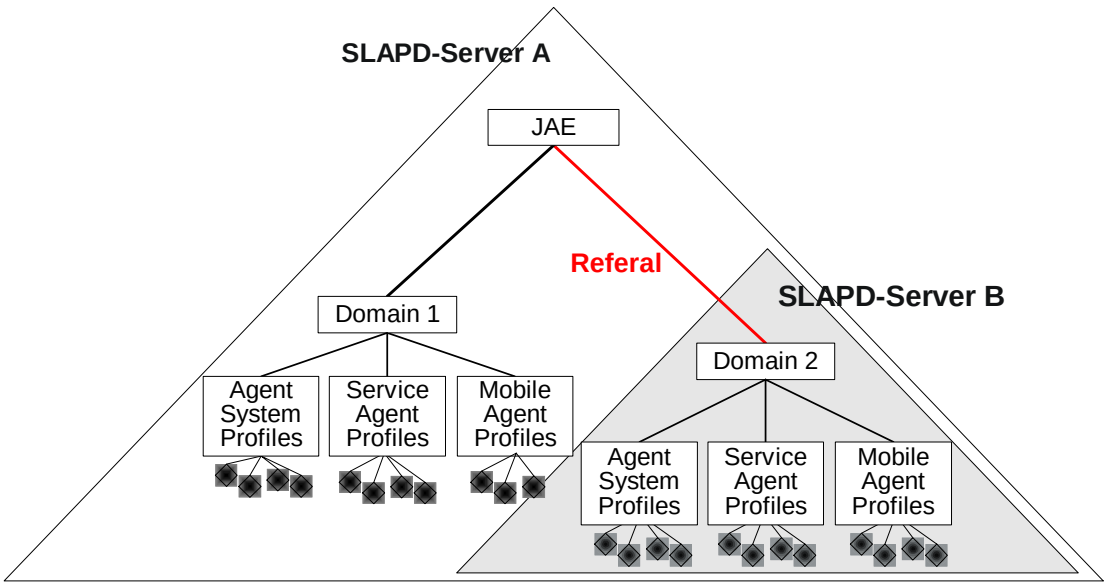

Figure 3-18: SLAPD Tree: Data Storage of Services, Agent Systems, and Mobile Agents

The common knowledge base about all available service agents is implemented in the agent directory: a directory service containing the service description of all service agents, migrating agents, and the agent systems. The structure of the agent directory realized with SLAPD is a hierarchical tree reflecting political, geographic or organizational boundaries. As shown in figure 3-18 different sub-trees represent different domains of the JAE network. Since SLAPD is a distributed database, different parts of the tree can be stored on various locations, but remain transparent to the clients. Even if there is only a small part of the tree stored in one location, a local client is able to access the whole LDAP tree. Search operations are automatically forwarded to other parts of the tree stored on distant locations. Therefore, every service center has access to all service descriptions stored in any part of the dictionary tree.

In JAE, three information profiles have to be stored, which are the information about the agent system, the service agent, and the mobile agents. Information look up is oriented towards these objects and therefore arranged in three domains below the sub-directory root as shown in figure 3-18 and labelled *agent system profile*, *mobile agent profile*, and *service agent profile*. The advantage of this directory structure is that information can be accessed by one object domain. Since information is distributed over several domains, but service description of a mobile agent can be found in each service profile domain, there is no need for a cost intensive search in different sub-domains. In the same way, modifications to the mobile agent profiles can be accomplished efficiently.

3.2.4 Measurements

In this section, some measurements are shown resulting from different trader approaches and varying number of requests initiated to the service center. Two concrete implementations with different combinations of the service center and the agent directory have been considered. The default agent system of JAE includes the service center with trading function in each agent system. JAE shows concepts similar to the FIPA's Directory Facilitator, but additionally exports the agent services to the central database - the agent directory - to export services to all other agent systems in the agent network.

The second version of the service center is designated “central” as the trading component is separated from the service center and located at the same host as the agent directory. The protocol used to access the service center mainly reflects the service center API, so that the direct comparison is more valuable. The OMG MAFFinder suggests this centralized architecture of the trader. It is based on the CORBA naming service with the MAFFinder as a central component on top of the underlying *Object Request Broker* (ORB).

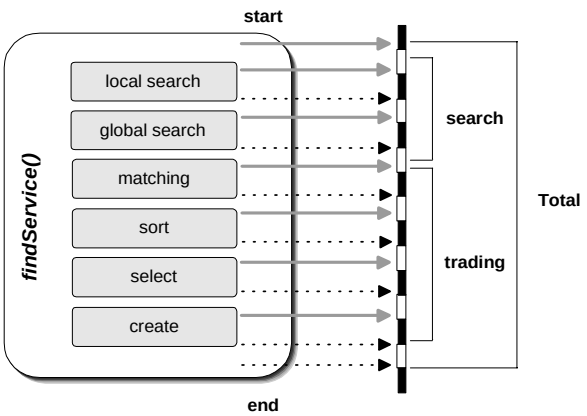

Figure 3-19: Covered Measurements of the look up Method

Figure 3-19 shows the measurement time-stamps for the *findService()* method. It reflects the single steps of the process described above. Obviously, the total service time is slightly dis-

torted due to the time-stamps, but the overhead of about 10 ms can be neglected. To simulate the access load of the service center and the agent directory, several mobile agents have been deployed in parallel, accessing the service center with different inter arrival times.

In the following, the central to the distributed trading approach is compared. However, the SLAPD cannot be modified to execute the trading functions of the service center. Thus, to overcome this problem, the above mentioned Java-based agent directory has been used for the comparable measurements. Earlier measurements have shown that the SLAPD is not very stable and that the Java-based agent directory is faster than the SLAPD. Figure 3-20 shows that the SLAPD server has an increasing service time, beginning with an inter arrival time of 0.3 seconds. The measurements shown here have been carried out on a Sun Ultra 143 MHz and 62 MB RAM. However, there is no significant change when switching from Sun Ultra to a Pentium 200 MHz and 64 MB RAM. Measurements have shown that up to 5 requests per second, the Java-based solution serves all requests with an almost constant service rate of about 400, ms. Only at higher arrival rates does the UNIX machine outperform the PC.

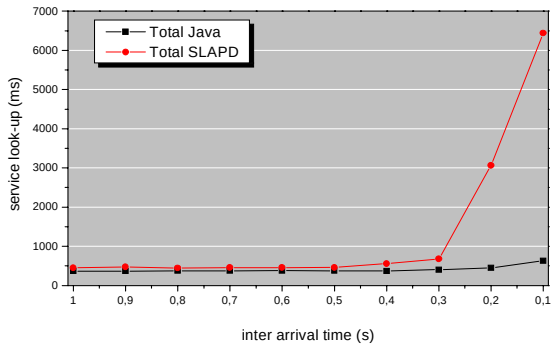

Figure 3-20: Service Center look up Time for SLAPD and Java Agent Directory

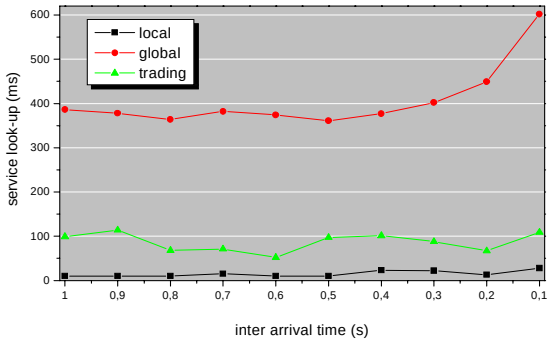

Figure 3-21: Measurement of the Java based Agent Directory

The remaining measurements have been done on identical PCs in a 100 Mbit/s switched Ethernet LAN. The relation between local and global service search and trading (when requested by one agent system) can be seen in figure 3-21. Trading and local search only form a small part of the overall look up time. About 3% are needed for the local search and 17% for the trading, the rest is consumed by the request passing through the network. This time varies depending on the network's load and size. Variations of the trading time occur due to the different number of resulting objects. For up to 5 requests per second, the only bottleneck is the establishment of the network connection and the protocol exchange between the agent system and the Java based agent directory. Only at an inter arrival time of 0.2 seconds, the total look up time increases due to the overload of the agent directory.

In the following, the effect of simultaneous access of the service center on the agent directory is discussed. The measurements have been gathered by increasing the mobile agents' requests

every 0.1 seconds. As could be expected, the service time of the service center increases at an earlier request rate due to the bottleneck of the agent directory and the parallel requests. Figure 3-22 and figure 3-23 show the average look up time for two and three systems respectively. The general shape of the curves remains similar due to the identical hosts of the agent systems. Up to an inter arrival time of 0.3 seconds, the service time is nearly constant at 500 ms, and thus about 25% higher than for single requests shown in figure 3-21. The increase in service time already starts at 2.5 requests per second and is more extreme. In contrast to the single service center measurement that embodies an ascent of 200%, simultaneous access of 2 service centers shows an increase of 700%. The diagram for 3 parallel agent systems is also reasonable with the average service time of the service center at about 800-900 ms with the steep ascent occurring for the inter arrival time of 0.5 seconds. However, a strange behaviour can be observed for an arrival rate of 5 requests per second. Here, the agent directory access decreases by about 2 seconds for each of the distributed service center accesses. One plausible reason could be a temporarily better performance of the network, but it is more reasonable to assume that the synchronisation of the parallel access has failed at that time. Of course, using more powerful machines and database programs can shift the increase to a higher request rate. Now it is interesting to discover whether the distribution of the trader really makes sense.

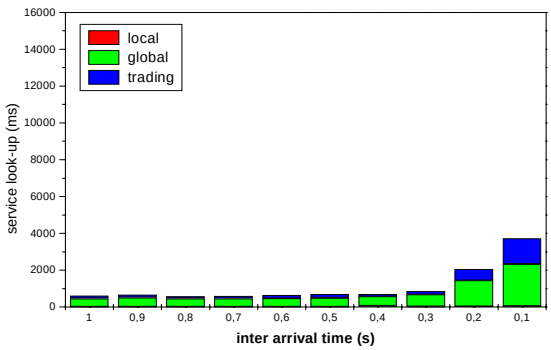

Figure 3-22: Look up time of 2 Agent Systems simultaneously accessing the distributed Trading Approach

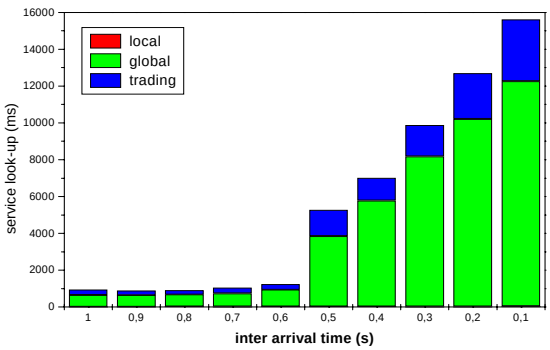

Figure 3-23: Look up time of 3 Agent Systems simultaneously accessing the distributed Trading Approach

The above measurements suggest that the central trading on one dedicated host may lead to additional overload. The distributed service centers are freed from the trading process, but with only about a 17% share of the overall time, this should be only a minor argument in favour of the central trading. On the other hand, the process time is minimal and the bottleneck is unambiguously the network, so that the central agent directory, including the trading, may not affect the total time. For the central trading approach, the trading process of the service center has been copied into the agent directory. The last step of the trading process still remains on the service center of each system due to security reasons.

In the following, the diagrams only show the total service time because of the synchronisation problems in distributed systems. Already the first comparison (figure 3-24) shows that the central trader approach has more problems at the rate of 5 requests per second than the distributed one. All other requests are served constantly in about 400 ms due to the fact that the minimum network connection time is the bottleneck. However, the less powerful machine, compared to the hosts of the agent systems that run the agent directory, causes the extreme ascent. The trading process takes longer than on the agent systems' machines.

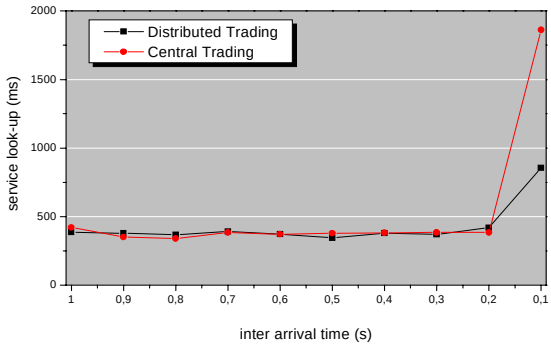

Figure 3-24: Comparison of the distributed and central Trading Approach

Thus, it is more interesting to compare the parallel access of two or three service centers. In figure 3-25 and figure 3-26, one can see the impact of parallel agent directory access for both distributed and central trading.

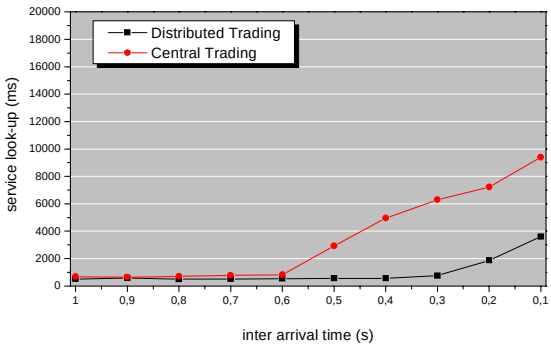

Figure 3-25: Look up time of 2 Agent Systems simultaneously accessing the distributed and central Trading Approach

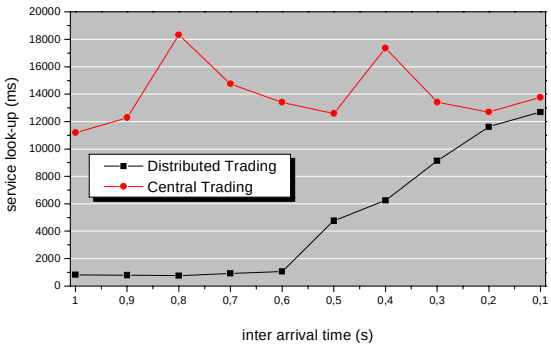

Figure 3-26: Look up time of 3 Agent Systems simultaneously accessing the distributed and central Trading Approach

The initial assumption has been confirmed by these simple measurements; the distributed trading approach is more effective. The diagrams show that the distributed approach allows a much higher throughput of service requests than the central approach. The extreme variations in the agent directory serving time are caused by the simultaneous access of the distributed systems. In worst case situations, all systems request the agent directory at the same time. In the

best case, all systems request the agent directory one after the other. Although the same effect is valid for the distributed approach, the lower working capacity of the agent directory without the trading results in a smoother curve.

The diploma thesis [Emm98] written during the JAE project contains further details and describes an approach for the mathematical modelling of the service center. The used birth-death queuing system [Kle75] for the performance evaluation is a simple first step to analyse the service center, but is not sufficient to take into account the parallel processes which influence the calculation of the service center. The research area for the analytical evaluation of service based agent networks is an open field that has space for further research. An analysing evaluation tool e.g. could help in agent based network planing, analysing and optimization. Concluding, the measurement of the implementation has shown that an architecture with a distributed service center and an agent directory is a well operating service trader for mobile agent systems. The measurements and the demonstration examples have been published internationally [PaLi98, PES98] and emphasize the results and confirm the proposed architecture.

3.3 Agent Migration in JAE

The mobility of code is always accompanied by security issues. Therefore, the solution and implementation of the agent migration in JAE is discussed, taking into account the security problems. Depending on the origin of a threat, the attack is caused internally or externally. Figure 3-27 depicts the potential attacks to agent systems affected by internal and external causes, respectively.

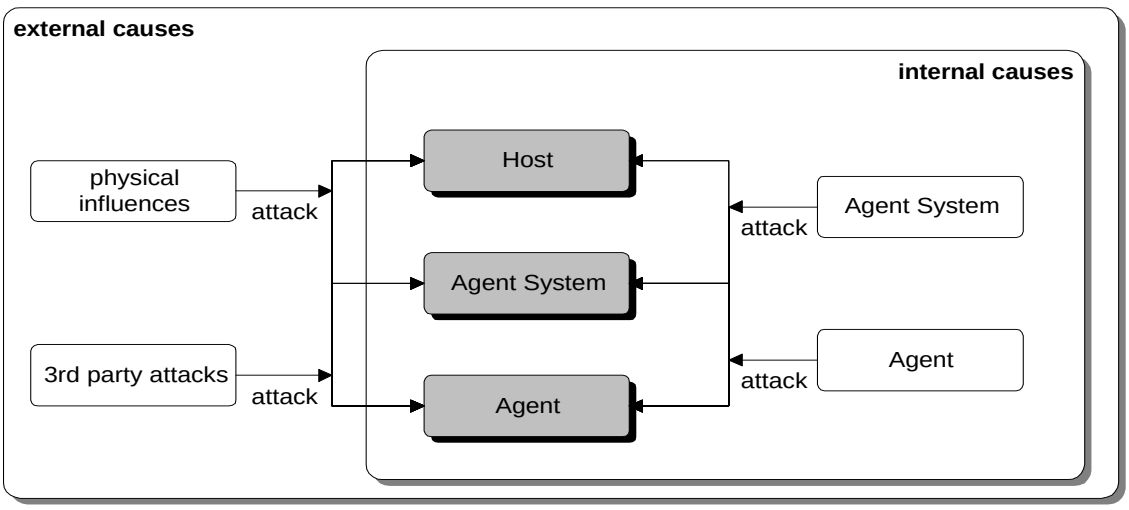

Figure 3-27: Overview of potential Attacks to an Agent System, an Agent, or a Host

External attacks to an agent system are distinguished by physical influences and 3rd party attacks. Internal attacks to an agent system, the agents or the host can be driven by the agent systems or the agents. In this section, only agent migration relevant security aspects will be regarded in detail. In case of problems concerning security matters by means of 3rd party attacks, the agent system relies on the security mechanism of the operation systems, because it is started with the restrictions of the user or the administrator of the agent system. The discussion of the internal attacks caused by hostile agent systems or agents in general goes beyond the scope of this section and therefore, the reader is referred to another diploma thesis [Swe98] of the JAE project. This thesis examines protection possibilities against hostile agents and agent systems. In particular, security solutions for JAE have been discussed during the work of the diploma thesis and parts of the results have been included in the system architecture of JAE.

In agent migration, the total loss of the mobile agent code by means of physical failures of the host or the network is fatal. Therefore, physical failures have to be considered by the ATP treatment. In JAE, a secure migration is established by an advanced ATP and its handling processes and the application of persistence. Persistence in this context means that the agent code has to be stored not only in the memory, but permanently on a long-lasting memory, e.g. hard disk. The right moment and the frequency to store the state of the mobile code, the data, and the code is crucial for a correct recovery. JAE is not a full persistent agent system as described by the thesis of B. Mathiske [Mat96], but secures the agent migration with the help of safety-catches. The agent manager stores all necessary data right before a migration and the ATP handler takes care of a persistent code storage before executing an agent on an agent system. Additional updates of the code on the persistent memory can be initiated by the agents, e.g. when important parts of a solution have been calculated. Considering a secure migration, not only the loss of mobile code during the transaction have to be prevented, but the execution of duplicated agents as well. The receiver of an agent only executes the agent if the transaction is successfully finished and the sender has committed the deletion of the agent on it's system, otherwise two versions of the agent could be in execution. The implementation details in the next section will show how duplicates are prevented and what secure migration means.

3.3.1 The Agent Transport Protocol

In JAE, the *Agent Transfer Protocol* (ATP) is not only an application protocol, but partly considers the particular nature of mobile agents. For example, a large amount of mobile agents might migrate to one destination system which can cause an overload situation, especially on small devices. To avoid this kind of situation, the system's Agent Transfer Protocol has to be enhanced by a mechanism. In JAE, the ATP ensures that each mobile agent is forced to send a so-called *Ticket* (see figure 3-28) to the destination system prior to migration. The destination system can check the available resources and send a reply to the mobile agent, indicating whether or not a migration is allowed. Thus, with the help of the ATP, the Security Manager,

and the agent manager, the destination system can avoid e.g. these overload situations. Before discussing the migration protocol of JAE, the content of the Ticket is presented in detail, because it is key to the secure migration concept of the ATP.

The first part of the so-called Ticket contains information about the agent transfer properties, the identity of the user and the originating system, and finally the migration information. The ATP properties allow setting the priority of migration of the mobile agent, the maximum number of connection retries before exception handling, and the delay of the reconnection attempts. The identity details are necessary in the first request for connection of the ATP. Depending on the identity, the general access permission of the destination host is given. Thus, mobile agents from certain users and systems can be simply denied. In JAE, the trust model for mobile agent is transitive, i.e. the trust domain of system 1, 2 and 3 is the same if e.g. system 1 trust system 2 and system 2 trusts system 3.

The transfer information includes: the destination, i.e. which agent system to migrate to, the current agent system the agent wants to leave, the departure time - which defines the preferred departure time of the mobile agent - and last but not least the origin of the mobile agent, i.e. the device where the agent started its life cycle.

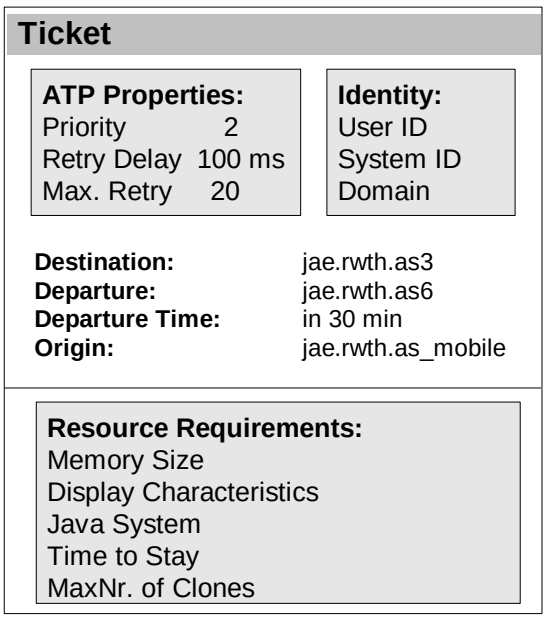

Figure 3-28: Ticket Object

The second part of the Ticket, containing the resource requirements, is delivered as soon as permission is granted from the destination host. As a second step during the migration negotiation, the resource requirements (examples are given in figure 3-28) are checked at the destination host. Depending on the Security Manager and the resource request, a migration permission or rejection is sent to the requesting agent system. For example, to avoid overload situations at

the destination device, a mobile agent is only accepted if the maximum of acceptable number of active agents is not exceeded.

The ATP is a multi-handshake protocol that ensures to save network resources and takes into account packet based bearer services such as GPRS. The reason for this is that because packet based bearer services will be charged per packet, the described ATP will reduce the costs. Contrary to an immediate agent migration - which in the worst case could mean twice the migration - because the mobile agent is not accepted at the destination host, a prior purification is wise, for example in wireless access networks. The mobile agent migration is completed when the agent's code and its state have been stored persistently in the destination system and both code and program of the agent were deleted successfully from the persistent database of the sender's agent system.

The multiple handshakes make sense to avoid unnecessary data transfer and for reasons of security and persistency. It would be a disaster if a failure of the ATP led to multiple copies of a mobile agent being executed on both the departure system initiating the exception handling, and on the destination system. In case of an error, the ATP can be reset to any of the numbered points in figure 3-29. Each successful transaction is recorded so that a repeated transfer after an interruption is avoided. Especially in wireless access networks, this is a very important feature that is not and cannot be covered by any underlying protocol layers.

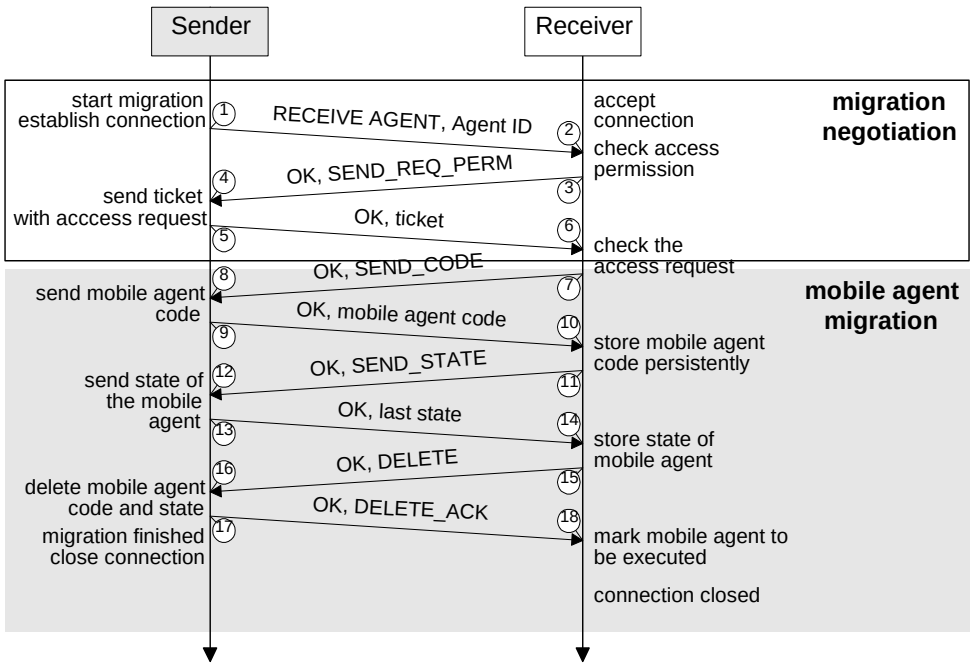

Figure 3-29: Negotiation Process of the Agent Transfer Protocol (ATP)

The ATP is divided into the negotiation and the agent migration phase and all together needs a minimum of nine interactions for a secure migration of a mobile agent. Nevertheless, the complex protocol can be managed with nine commands as depicted in Table 3-30.

Command	Value	**Description**
PING		test ATP connection
SEND REQUESTED PERMISSION		request agent's Ticket
SEND CODE		request agent's code
SEND STATE		request agent's last state
DELETE		delete agent on the sender
DELETE ACKNOWLEDGED		acknowledge deletion
RECEIVE AGENT	Agent ID	migration initiating tag
OK	Command / Data	correct protocol tag
ERROR	Error Message	failure protocol tag

Table 3-30: Agent Transfer Protocol Commands

The protocol is not stateless and therefore, needs more intelligence in the ATP handler, because depending on the request, the value of the OK command can be data (code, state, Ticket) or a command. Consequently, the achievement of a secure migration is dependent on the interplay of the agent manager, the ATP handler of the communication manager and the persistence storage of the JAE agent system. The interplay of these modules guarantees automatic safety-checks, control of the number of executed agents, persistent storage of agent codes, and parallel handling of migrating agents.

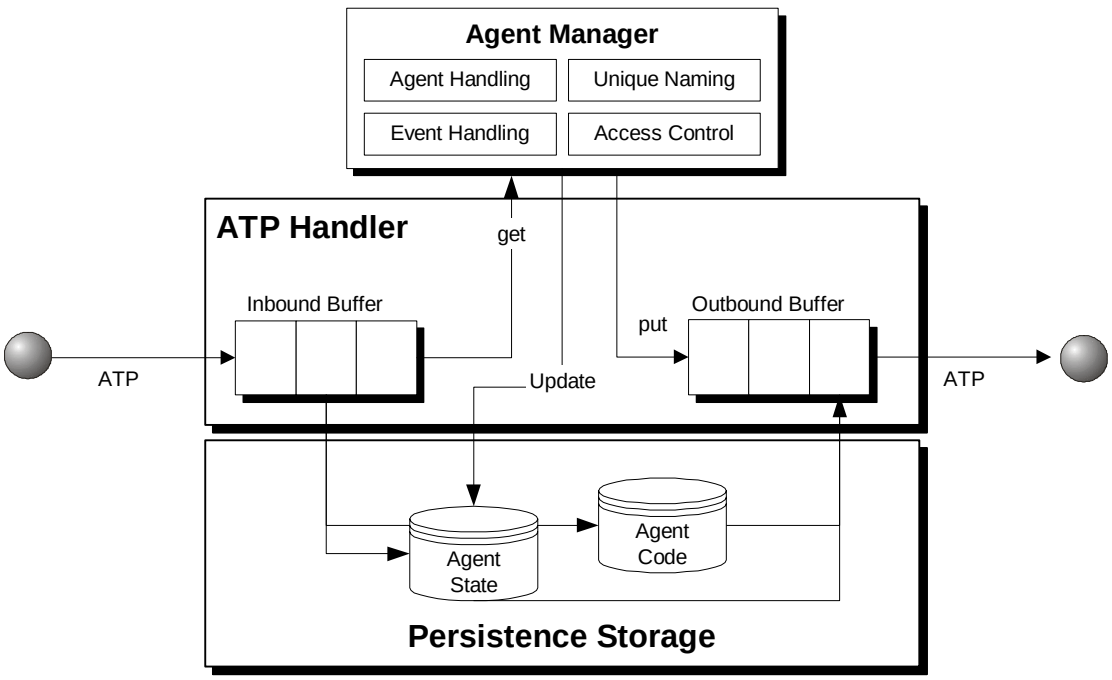

Figure 3-31: Interplay of Agent Manager, Persistence Storage, and ATP Handler

As shown in figure 3-31, the agent manager monitors and controls activities of the mobile agents on the one hand, but the ATP handler transparently takes care of the migration on the other hand. In the overview figure of the ATP handler, two buffers are identified, which are in close dependence to the persistence storage, namely, the *agent state* and *code storage*. The inbound buffer of the ATP handler helps to optimize the agent resource planning, i.e. depending on the number of parallel executed agents in the system. New incoming agents are only stored in the persistence storage, but are not executed by the agent manager. In a similar way, the outbound buffer supports to reduce computation power. The inbound and outbound buffer size and the maximum number of agents executed in parallel in the agent system are determined in the configuration file of the agent system. This configuration file can be edited and initiates the system on re-booting.

In JAE, the Ticket concept and the ATP handler allow agents willing to migrate to store in the persistence storage and to free the memory of the agent system. Thus, other agents can be activated or agents departing at a later moment or waiting for the destination agent system (on a mobile device) to be switched on do not consume precious resources. The agent's code is stored in the usual *Java Archive* (JAR) file and the state is stored with the JDK serialization class. In the current JAE version, the persistence storage is implemented as a directory to keep the system requirements for the JAE agents system low. It is recommended, due to database transaction safety and additional security aspects of a database, to use them as persistence storage in later versions.

Figure 3-31 also shows that mobile agents are stored automatically by the ATP handler on arrival at the agent system and before migration. The decision to have two safety-catches of the agent's state has been made to reduce calculation time on re-execution due to migration failure. In the worst case of a migration process, the agent's state and results are lost and the complete calculation of the agent has to be repeated. Independent of the two safety-catches, the update of the agent state can be set by the agent programmer any time, e.g. after a problem solving.

A critical phase of the ATP is constituted when the link is not properly disconnected after the check-point 14 in figure 3-29. In this end phase of the protocol, a doubling of a mobile agent can occur, because a completely transferred and executable agent is available in the new system and the originating system has not deleted the agent at this moment. Here, a definite solution is necessary for the sender and receiver ATP handler. Table 3-32 gives an overview of the problem treatment of both sender and receiver ATP handler on failure at different points in time.

Fundamentally, the mobile agent on a new location can only be started if the acknowledgement of the deletion has been received, but what happens when a network failure or an agent system malfunction leads to an undefined status on both the sender and the receiver system? Consider the situation when the sender has deleted the agent code (status after check-point 16) and the link has been disconnected during acknowledgement of the deletion. There is no problem if the

Failure during check-points (figure 3-29)	Action at Sender ATP Handler	Action at Receiver ATP Handler
1- 14	retry of connection depending on Ticket parameters and alternative treatment of agent if migration is not possible	interruption of the ATP
15-18	re-start of the mobile agent after certain time-out and manipulation of the agent state for exceptional migration handling	after certain time-out, the failure information is saved by means of the maintenance concept (chapter 4.3.2)

Table 3-32: ATP Failure Treatment of Sender and Receiver on existing Disconnection

connection can be re-established, after another handshake the situation is clear to both parties. The undefined status has to be solved when the connection cannot be established again. At this point, the ATP handler, sender as well as receiver works with a time-out: If the connection has been interrupted after check-point 15, the sender can re-start the mobile agent on its system after a time-out, because without the deletion acknowledgement the execution on the receiver is not allowed. On re-start, the mobile agent's state is manipulated by the agent system to be treated exceptionally. The developer of the mobile agent might change the actions by defining the `HandleMigrationException()`. The default setting for this case is the return of the mobile agent to its origin. If the connection has been interrupted after check-point 16, the re-start of the agent on the sender is not possible. In both cases, the receiver has to report this failure information after a certain time-out. In JAE, the maintenance concept has been implemented to have a safe keeping for this kind of failure information. The details of the maintenance concept are discussed in chapter 4.3.2.

3.3.2 Implementation Details

An optimized and secure migration has been one major goal of the implementation in JAE. It took a lot of effort to discuss, design, and implement the ATP and the corresponding modules, which led to safety-catches, parallel treatment of incoming and outgoing mobile agents, and a Ticket concept with priority settings to optimize agent migration. Having the ATP negotiation in mind (figure 3-29), it is necessary to have a closer look into the ATP handler.

For optimization purposes, a pair of inbound and outbound ATP handler establish the agent migration link. Figure 3-33 shows the implementation details of an outbound ATP handler with 2 buffers, which on the one hand allow multiple, parallel ATP connections and on the other hand, consider priority concepts introduced by the Ticket queue. The inbound ATP handler is built analogous to this case.

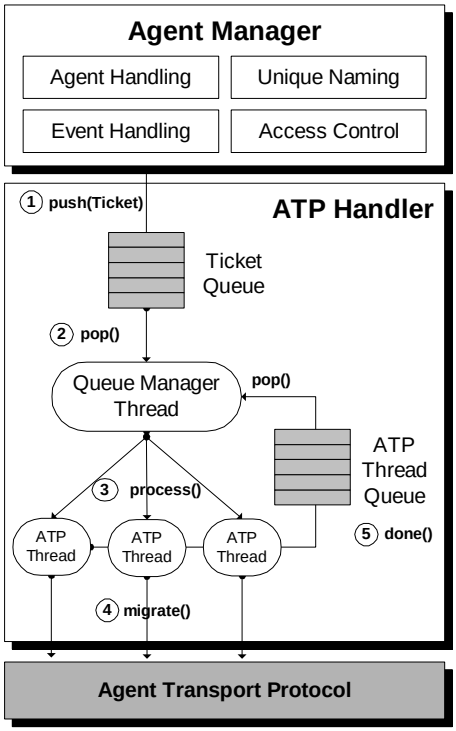

Figure 3-33: Implementation Details of the outbound ATP Handler

After the persistence storage of the code, requests for migration are done by queuing a Ticket into the ATP handler. The Ticket queue is sorted by departure time and its priority. It is the *queue manager thread* taking the next Ticket in the Ticket queue and allocating the next free *ATP thread* from the *ATP thread queue* for agent migration purpose. The ATP thread takes care of the safe migration process as described before. This process is repeated as long as the Ticket queue is filled and the departure time of the agents matches. The size of the queues are variable and are set by the agent system parameters during a re-start of the agent system. The ATP thread keeps track of the protocol status and treats the failures. Some information from the Tickets are used by the ATP thread, e.g. the retries and time outs are monitored by the thread, if re-scheduling of an agent - due to migration failure or some time-out - has to be handled, a simple re-queuing of the corresponding Ticket in the Ticket queue of the ATP handler is sufficient.

Finally, the whole migration process is considered in detail in a sequence diagram, which is a slight modification of the *Unified Modelling Language* (UML). In figure 3-34, the process starts with the execution of an agent. With the help of the service center, the location of the requested service becomes available to the mobile agent. The returned Ticket object allows the start of the migration. The *go()* method causes the migration permission check and, on success, initiates the serialization of the mobile code and the persistence storage of the agent. The Ticket is handed over to the ATP handler and the mobile agent is deleted from the memory. If necessary, the restart of the agent occurs with the state information and the program code from

the persistence storage of the hard disc or secondary memory, and the Ticket in the ATP handler queue. The scheduler of the queue manager thread of the ATP handler initiates the migration of the mobile agent with the help of the ATP threads and the Agent Transfer Protocol as described before.

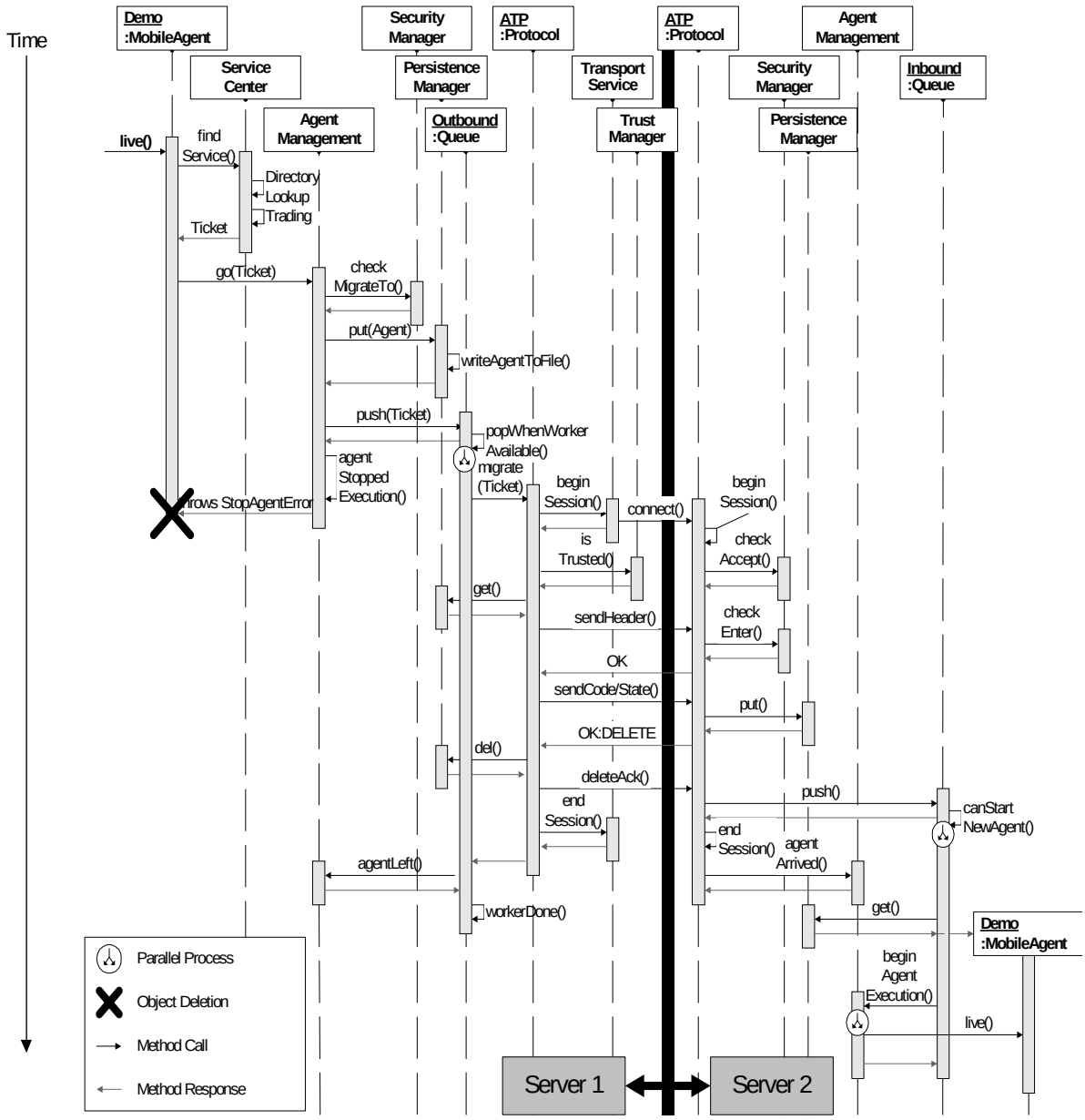

Figure 3-34: Migration Sequence Diagram

The separation of the physically distributed agent servers is clarified in the figure with a thicker line. The notation of the UML leads to the assumption that the communication between the two servers is realized by remote procedure calls or remote method invocations, but in this case, it is the ATP application protocol. The receiving agent server stores the state and the code of the mobile agent in the persistence storage and the Ticket in the Inbound Buffer of the ATP

handler. Analogous to the sender agent server, the mobile agent can be started on the new location with the information given by the Ticket and the mobile code. The execution of the mobile agent on the new server may depend on the number of already running agents. With the final acknowledgement of the agent manager, the migration process is finished and the execution of the mobile agent is indicated outermost right in the figure.

Already today a simple two step protocol similar to the ticket concept is successfully used to download Java applications to a Java runtime environment in resource-constrained hand-held devices so called *Mobile Information Devices* (MIDs), e.g. mobile phones and PDAs. The *Mobile Information Device Profile* (MIDP) in its current version 1.0 specifies the so called *MIDlet* - the application based on the *Java 2 Micro Edition* (J2ME, [CLDC02]). MIDlets can be downloaded from the Internet over any wire line and wireless (e.g. WAP) connection. One or more MIDlets are packaged together into what is referred to as a *MIDlet suite*, which is basically a standard JAR (Java archive) file. The JAR file includes a *manifest* with a number of MIDP-specific entries that describe the MIDlets in the suite. A separate file, the *application descriptor* (*JAD* file), contains similar information as the manifest, and is used by devices to obtain information about a MIDlet suite without having to download and install the MIDlet suite first. The JAD file can be seen as a simple version of the JAE Ticket. The example of an application descriptor (JAD file) for a MIDlet is shown in Program 3-35 and contains information about included classes, any other resources (such as images), the Jar file size, the URL, the name, the vendor and the version.

```
01 MIDlet-1: MyMIDlet, MyMIDlet.png, com.developer.j2me.MyMIDlet
02 MIDlet-Jar-Size: 4238
03 MIDlet-Jar-URL: http://www.rwth-aachen.de/MyMIDlet.jar
04 MIDlet-Name: MyMIDlet
05 MIDlet-Vendor: RWTH Aachen
06 MIDlet-Version: 1.0
```

Program 3-35: MIDlet Application Descriptor (JAD file)

The JAD initiated installation process shows that in comparison to the ticket concept and the ATP the interaction with the user is required. The installation of a MIDlet is initiated by the user selecting a JAD file. This may happen by the user clicking on the JAD in a web or WAP browser, or on an email attachment or clicking on a JAD file supplied to the mobile phone using some other transfer protocol, e.g. via infra-red, Bluetooth, WAP, SMS, HTTP etc. In any case the so called *Recognizer* ensures that the *Installer* is started when presented with a JAD file. The JAD file is downloaded and parsed for the information needed for installation by following steps:

1. Examination of the MicroEdition-Configuration and MicroEdition-Profile attributes. The versions required by the suite are compared with the versions supported, and if unsuitable the user is notified and the Installer does not proceed.

2. Displaying details (JAD information) of the software to the user, requesting confirmation that the installation should continue.

3. Comparison of the MIDlet name, version, and vendor with the corresponding attributes of all the MIDlet suites installed on the device. If the MIDlet name and vendor are the same as a suite already installed, the user is prompted whether to overwrite the existing version.

4. Interaction with the user giving the choice to install the MIDlet and the option to proceed with, or cancel, the installation. The attributes MIDlet-Jar-Size, and data size are used to display information to the user on the required amount of disk space.

5. Examination of the MIDlet-Jar-URL attribute and if the scheme specific part of the URL is not supported (i.e. is not HTTP) then the user is notified, with a dialogue displaying an invalid download location, and the Installer does not proceed.

6. Download of the JAR file from the given URL.

7. Checking the manifest of the downloaded JAR with the values of the MIDlet-Name, version, and vendor attributes of those of the JAD. If they are not identical, the user is notified, the JAR and JAD files are deleted from the phone and the Installer does not proceed.

In comparison to the ticket concept most of the decisions are left to the user. The user has to decide whether there is enough memory space to install the application and whether the installation should be continued or cancelled in general. Currently, MIDlets are the only successful *Over The Air* (OTA) mobile application for mobile devices and are outstandingly well accepted in the mass market for mobile games. The replacement of the current MIDP version 1.0 (released in September 2000) with version 2.0 in 2003 shows on the one hand that there are some lacks in the specification, but on the other that there is a wide acceptance of this specification (most of the latest mobile devices support MIDP) and the improvement of the MIDP specification is continued [MIDP02].

3.3.3 ATP Measurements

The protocol with JAE has been successfully tested, interrupting the ATP during any of the transactions by suddenly stopping the agent system. Obviously, this migration protocol has an overhead, as can be derived from the measurements given in figure 3-36. The comparison of the mean values of the ATP with just the Java stream transmission of code and state show that the persistent ATP requires twice as much time. The transfer time becomes much worse, using the *Secure Socket Layer* (SSL, [FKK96]). The measurements of agent migration were repeated about 200 times between Sun Micro Systems Ultra1 3D work-station with 128 MB memory running Solaris 2.6 and JDK 1.2. The size of the mobile agent was 11466 bytes consisting of 3 classes in a JAR file (9030 bytes Byte-code and 2436 bytes serialized data). The local area network was a 10 Mbps Ethernet.

To give an impression of the network performance, here are a few values: the transmission of 5 Mbytes via FTP had a throughput of 650 Kbps, and the round trip of a ping packet (32 bytes)

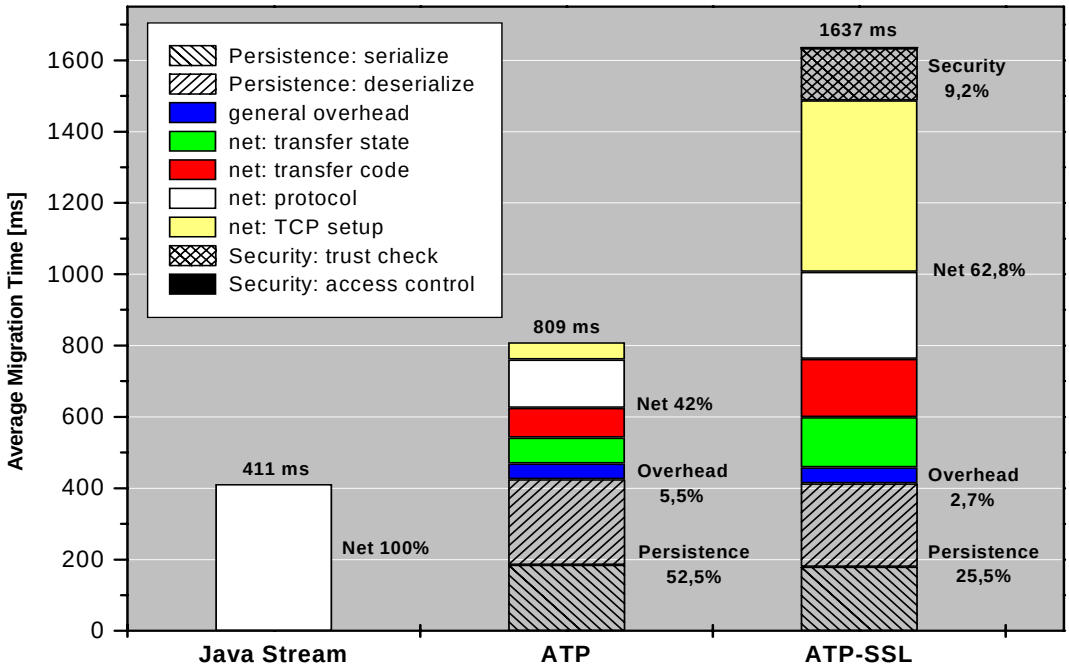

Figure 3-36: Comparison of Agent Transfer Protocols

was less than 10 ms. Finally, the comparison of the migration time has shown that the expectation has been proven right: for protocol security and code persistency, a high performance price has to be paid.

3.4 Agent Communication in JAE

The communication between agents is an important achievement that has been topic to research since the early seventies [ErLe75]. With the mobility of code additional problems appeared and have been the major research subject in several agent systems, not only in implementations but theoretical researches [RBM00].

There are various inter-agent communication concepts realised in agent systems. In this thesis only the two communication concepts which have been implemented in JAE are briefly introduced. First the blackboard communication and second the message exchange communication. Both communication concepts in combination allow various, competitive agent-to-agent and agent-to-group communication. In JAE the communication between heterogeneous agent systems, i.e. language and system independency has been an important requirement. Therefore, the ACP handler of JAE supports HTTP as one major Internet application protocol.

3.4.1 Blackboard System

In the research of Artificial Intelligence the so called blackboard system has been of crucial importance. A first application based on a blackboard system has been the speech recognition program HEARSAY-II [EHLR80] that has been developed during 1971-1976 by *Defense Advanced Research Projects Agency* (DARPA/ARPA). In some early multi agent systems such as the *M* system of integrated agents [Rie94] and also in mobile agent systems such as ffMain and Siemens Swarm the blackboard has been implemented as a local communication system.

The most important variant of the blackboard systems is based on *Tuplespaces*. Tuplespaces became popular at the University of Yale when David Gelernter invented the language extension LINDA to communicate between parallel processes and parallel computing environments [ACG86]. The LINDA concept fits very well in distributed systems. The implementation of the language extension into JAVA underlines the importance of this simple but powerful concept. JAVA Tuplespaces improve the success of blackboard systems also in wide area distributed applications such as the service network in the Internet.

Blackboards based on Tuplespaces only permit tuples as entries. Tuples are sequences of a specific data type. The example sequence (“hello”, 5, 12.5) is a 3-tuple consisting of the types *String*, *Integer*, and *Double*. In this context templates are tuples with place holders for certain types of variables. Templates that are matching tuples have the same number of elements in their sequences with the same type at each sequence position, e.g. (True, 5.5, “Otto”) consists of the types *Boolean*, *Double* and *String*. This tuple matches with itself and the template (*Boolean* B, 5.5, *String* S), but not with (*Boolean* B, *Integer* X, “Otto”) or (*Boolean* B, 5.5, *String* S, “and another String”). Substantial relations between tuples and templates are given by wild cards which make Tuplespaces more powerful: for example the simple 2-tuple (True, *) matches with (True, 5.5) as well as (True, “Otto”) and any other tuples and templates which starts with a *Boolean* type and have two entries.

The operations on Tuplespace defined by LINDA are extensions that can be added to any programming language. Originally LINDA operations have been used to communicate between parallel processes with the distinguishing feature of anonymous and asynchronous message exchange. In distributed systems this is a valuable feature allowing time and spatial independent communication between distributed applications. The set of commands to interact within a Tuplespace is small and essentially consists of: *out()*, *in()*, *read()*, *write()*, and *eval()*. The data types (*Integer, Long, Double* etc.) are depending on the language that LINDA is extending.

The *out()* command stores a tuple in the Tuplespace, e.g. *out(5.5, “Otto”, -12)*. With the commands *in()* and *read()* a result from the Tuplespace can be retrieved. The *in(template)* command returns one matching tuple and deletes it from the Tuplespace, though the *read()* command returns one matching tuple but retains the matching tuple in the Tuplespace. Both operations only return one result that is randomly selected out of a set of matching tuples. The *eval()*

command allows to embed functions into the Tuplespace. This so called *active tuple* consists of data and functions, e.g. *eval(5.5, function(42), 12)*. This operation first calculates the content of the tuple and stores only the results of the functions.

The project *Mobile Agent Reactive Spaces* (MARS) from the University Modema, Italy, has its research focus on reactive Tuplespaces for agent communication purpose. In the MARS System an additional meta Tuplespace has been defined to support dedicated actions depending on the agent's operations. Within MARS the Tuplespace easily supports to filter tuples that has been added to the meta Tuplespace since the last access of the same agent [CLZ00]. A tuplespace project dealing also with the code mobility is described in [PiBu02]. The project deals uniformly with code and data. Tuples contain classes, and tuplespaces become the code base. Transient sharing allows for location transparent retrieval of classes, and accommodates changes determined by the re-configuration of the system, e.g., due to mobility. Further JAVA based implementations are the T Spaces from IBM [LCX+01], and the JavaSpaces from Sun [JS03]. The continuously development of the Java Tuplespace in the industry underlines the importance and relevance of the blackboard concept to distributed applications.

3.4.2 Message Passing

In most agent systems the message passing between agents is the first choice for communication. A substantial implementation of this communication concept can be found in the commercial Voyager Messaging that is based on the version 1.0.2 of the *Java Message Service* specification (JMS) from Sun Microsystems [Voy03]. It supports the 4 most important and necessary message patterns:

1. the fully synchronous communication
2. the one way non blocking communication when messages are dispatched concurrently
3. the time depending synchronous communication
4. the time depending asynchronous communication

An important requirement for the message passing communication in agent system is the unique addressing scheme. All communication forms either synchronous or asynchronous between two agents or a group of agents need the unique ID of an agent to find and address the mobile agents in the system. The handling of unicast, broadcast, or multicast communication (e.g. agent to group communication) even becomes more complex. A promising solution to address mobile agents and to keep up the communication has been introduced by means of badges in the MOLE agent system [BHR+98]. The badge concept is an interesting solution hiding the communication infrastructure.

In MOLE badges allow a semantic addressing of mobile agents, i.e. the attributes of a badge describes the task of an agent and even its property. The badges can be dynamically enhanced and deleted, e.g. an agent may have the badges with the attributes "*group MATRIX*", *"group password NEO"* and "*mobile*". The informal addressing of this agent could be described with the badge "*to all MATRIX group member having the password NEO and are mobile*". With the flexibility that comes with the badge solution the problem of semantic uniqueness raises. The clear meaning of words and sentences is crucial to the communication in general. For the communication between agents and to realize the badge concept matching each other a global list with keywords similar to the meta keywords has to be defined. A more specific consideration of agent communication with message passing is necessary due to the fact that mobile agents may change their location. There are several approaches beginning with mathematical computation of location probabilities of mobile agents up to paging solutions of the telecommunication. Common solutions to find and address mobile agents are:

1. Forwarding of messages means that messages are delivered according to the migration path of the mobile agent. The mobile agent leaves a trace at the visited agent systems describing the next destination. The agent system takes over the forwarding of messages and deleting of the trace. Unfortunately, this very simple solutions fails when the route is broken because of a failure or halt of an agent system. Furthermore, messages may be delivered too late because the agent has been terminated before.

2. Broadcasting is a very simple but resource expensive solution to localize an agent due to the fact that all agent systems in the agent environment have to be requested. The process is quite simple; all agent systems are requested to acknowledge the request within a specific time out when executing the mobile agent. The unique ID of the responding agent system allow the delivery of the message. Another proposed solution is the periodical broadcasting of the mobile agents' location to all systems. All agent systems have to update and synchronize the location of mobile agents in their local database and may deliver messages based on this information. Though, due to outdated location information the false routing of messages can be another problem. To reduce the message delivery to wrong destinations the departure and arrival system can be buffered, even so broadcasting into an open or large environment means a lot of overhead. A variant to the simple broadcasting can be the right choice, being the most effective and fast solution when the resources are given. In the telecommunication system paging for example is an efficient way to localize a caller. In a GSM mobile terminated call, the called mobile station has to be paged within a group of cells to determine the exact location/cell of the mobile station. This results in a call path being set up through the GSM network which represent an inefficient use of network resources but is an efficient and fast method [Paging01]. An in deep research in new localization and migration approaches for mobile objects can be found in Axel Küpper's dissertation [Küp01] and will not be discussed further in this work.

3. The inclusion of a Directory Service allows a more decentralized localization process. Each mobile agent has to be registered on arrival and de-registered on departure into a global directory service. With this automated process the directory service stores all current location information of the mobile agents. To address a mobile agent only a look up in the directory service is required.

Obviously, in JAE the message passing system has been implemented including the Agent Directory. Thus, the distinguishing feature in the agent communication of JAE is the mixture of the global blackboard system based on the LINDA concept and the messaging system based on the badge concept to support unicast, multicast, and broadcast communication.

3.4.3 Implementation Examples

Having the architecture of the JAE agent system in mind (figure 3-2) the communication manager of JAE is the only interface to the network for all agents in the agent system. Thus, in close relationship to the agent manager the two communication concepts are handled by means of the ACP handler. Registration, addressing, look up, and updating of mobile agent information are handled by the agent manager. All information are stored with the help of the ADP handler in the global Agent Directory and locally in the persistence storage. The figure 3-37 illustrates the modules involved in all agent communications.

The Tuplespace has been implemented as a Java library and thus can be integrated easily into any agent system or application. In JAE the blackboard communication can be used by simply integrating the Tuplespace library into a system agent and offering the Tuplespace as a global service. The Tuplespace operations for the clients are supported by the communication manager. The global Tuplespace itself can be placed anywhere in the environment by a system agent.

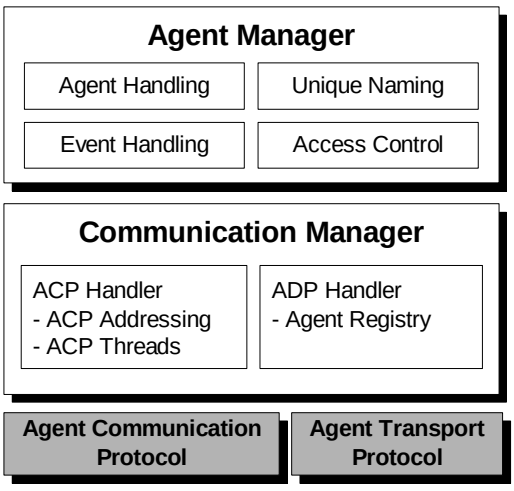

Figure 3-37: The Communication and Agent Manager

For the sake of simplicity only *String* tuples and templates have been implemented matching with the type *String* and *NULL*. In further implementation the support of other types will be considered. Following core tuple operations have been implemented in the prototype:

- *write(tuple)* - while saving the tuple into the Tuplespace the client is blocked
- *asyncWrite(tuple)* - while saving the tuple into the Tuplespace the client is not blocked
- *read(template)* - returns *Null* or one result
- *readAll(template)* - returns *Null* or all matching tuples
- *take(template)* - returns *Null* or one result and deletes the tuple from the Tuplespace
- *takeAll(template)* - returns *Null* or all matching tuples and deletes all tuples from the Tuplespace

The number of tuple elements is not restricted by the classes. The only limitation of the arguments is the object constructor that is an array of strings. The access to the blackboard is transparently supported by the communication manager of JAE and is based on the application protocol HTTP. Thus, different agent system take advantage of the standard protocol and any JAVA based agent system can be easily enhanced by the Tuplespace class. Figure 3-38 depicts the distribution of Tuplespace clients and the Tuplespace in the environment.

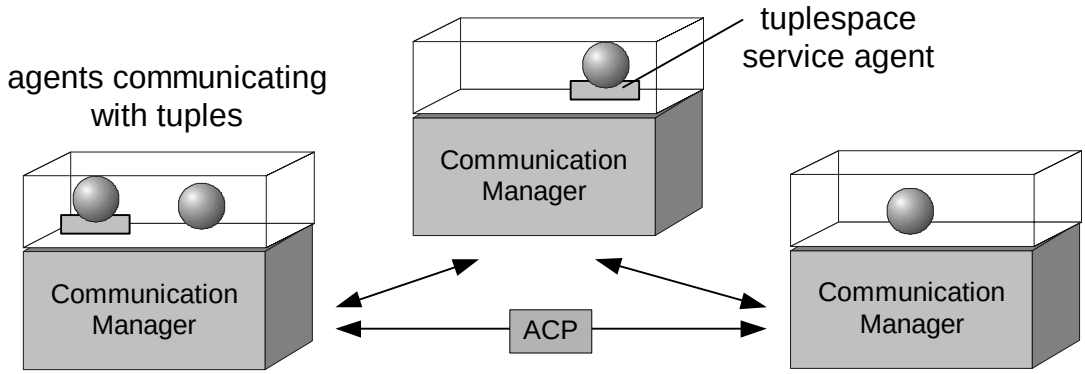

Figure 3-38: Diagram of the Tuplespace Communication in JAE

Alternative protocols that have been not implemented are IIOP or the *Simple Object Access Protocol* (SOAP, [SOAP00]). The latter language-neutral standard uses the *Extensible Markup Language* (XML, [XML03]) as its on-the-wire protocol and thus offers maximum flexibility for future enhancements. For further implementations SOAP is recommended as the blackboard communication protocol upon the supported HTTP.

The Program 3-39 gives an example how to define a *String* tuple and how to write this tuple into the Tuplespace. All other tuple and template methods can be applied accordingly. On the basis of this example the strength of this simple communication concept is easily comprehensible.

```
01 String arg = new String[4];
02 arg[0] = “Otto”;
03 arg[1] = “5.5”;
04 arg[2] = “42”;
05 StringTuple tuple= new StringTuple(arg);
06
07 result = tuplespace.write(tuple);
```

Program 3-39: Example of a Tuple Storage into the Tuplespace

As mentioned earlier the messaging system of JAE depends on the Agent Directory and is based on the badge concept. This combination allow synchronous and asynchronous as well as one way and group communication. The identification attributes within the badges are again meta keywords and its values. The most important meta keywords reserved for this addressing concept are given following:

- The *Global Unique IDentifier* (GUID) corresponds to the FIPA standard and is reserved within JAE by the meta keyword BADGE.GENERAL.GUID. There are a few more general information addressed by the two keywords BADGE.GENERAL such as the agent system ID by BADGE.GENERAL.SYSTEM or the used agent system language by BADGE.GENERAL.LANGUAGE.
- The information exchange between agent systems is defined by the meta keyword BADGE. added by the name of the agent system, e.g. BADGE.JAE.CLASSNAME.
- A group can be addressed by the meta keyword BADGE.GROUP. added by the group name, e.g. BADGE.GROUP.Matrix. The value can be used as group password or identification.

All other meta keywords are not reserved for the badge communication and can be defined depending on the context as described in the chapter 3.2. The following Program 3-40 shows an example that addresses all members of the group with the name “Matrix” and “Neo” as the group password within the agent system JAE and containing the freely defined group attribute “MOBILE”. The message is sent synchronously to all agents matching the criterion. The attributes of the send method are the reference to the initiating agent, the defined badge, and the message (line 6). The time out variable in the example defines the waiting time for replies and is blocking the agent until a result is given or the time out has been reached, returning *NULL* as a result. When necessary, the communication can be checked again after the time out with the *check()* method of the *GroupReply* class.

```
01 Badge badge= new Badge();
02 badge.put(“BADGE.GENERAL.SYSTEM”,”JAE”);
03 badge.put(“BADGE.GROUP.Matrix”,”Neo”);
04 badge.put(“MOBILE”);
05
06 GroupReply reply=myACP.sendBidirectional(this, badge, message);
07 Message answer1=reply.waitForResult(timeout);
08 Message answer2=reply.check();
```

Program 3-40: Badge Addressing Example

The communication manager transparently takes over the task to find the agent ID within the local database and the global Agent Directory. In the case of the above given example, the location of all agents within the “Matrix” group are requested. The ACP handler establishes all bidirectional connections with the help of the ACP threads and returns all replies to the initiating agent. The asynchronous message passing works in accordance to this example. Further details and measurements on the different communication concepts can be retrieved from the diploma thesis [Nes98] that has been part of the JAE project. The analysis of the measurements did not result in new important discoveries and corresponded to the expectations. The combination of the two communication concepts is promising and has to be proven by applications applied to a wider community.

3.5 Outlook on JAE and FIPA

The FIPA standard has become widely accepted and many implementations claim to be FIPA compliant. Thus, the question arises: which are the distinguishing features of JAE? The FIPA standard fully embraces the agent paradigm and in particular it defines the reference model of an agent platform and a set of services that should be provided. The collection of these services, and their standard interfaces, represents the normative rules that allow a society of agents to exist and operate. In spite of this, the standard targets interoperability and, as a consequence, it focuses on the external behaviour of the system component, leaving open the implementation details and the internal architecture. This is an important fact that results in unique internal architectures although the implementations are FIPA compliant.

The main focus of this thesis is in the architecture and the implementation of the prototype, demonstrating the service-based programming concept and the mobility support rather than being FIPA compliant. From the functional point of view JAE provides higher-level libraries to enable easier and more effective distributed application development by providing generic services and a common API within simple-to-use building blocks - being operating systems independent.

The architecture of JAE with its solutions for service trading, agent migration, and communication are unique in this constellation. There is not one agent system handling the service trading in this way or having implemented the agent transport protocol with the multiple handshake process including the ticket concept. Especially the ticket concept can not be found in this detail in FIPA. However, the main services required for a FIPA compliant architecture can be found in JAE and the adjustments to meet the external behaviour may be realized with a reasonable effort.

- The FIPA Agent Management System (AMS) corresponds to the JAE agent manager.
- The FIPA Directory Facilitator (DF) corresponds to the JAE service center.

- The FIPA Agent Communication Channel (ACC) corresponds to the JAE communication manager.

Much of the (FIPA) services comes with JAE and is hidden to the developer.

- There is no need to implement the agent system (FIPA Agent Platform). The agent manager (AMS), service center (DF), and communication manager (ACC) are launched automatically at start-up.
- There is no need to implement agent-management functions. The agent is registered with the agent system by the agent classes and it is given a unique name and an address.
- The agent classes provide simplified APIs to access the services of the service center (DF) for registration, searching, etc.
- There is no need to implement message transport and parsing methods. The agent system automatically (and possibly efficiently) takes over the tasks when sending/receiving messages.
- There is no need to implement interaction protocols. By means of the communication API only the events after receiving messages have to be described.

The Agent Communication Language (ACL) is one of the main assets of the FIPA standard. It is based on the speech act theory and on the assumptions and requirements of the agents' paradigm. FIPA standardized an extensible library of 22 communicative acts that allow representation of different communicative intentions such as requesting, proposing, informing querying, calling for a proposal, refusing, etc. ACL has been accepted as a standard by the agent community because of its detailed definitions on the structure of a message, the representation and conveyance of information, the content of the message and its properties, and the information to identify, follow and time-out threads of conversation between agents.

To become FIPA compliant the libraries for the ACL have to be included to the JAE agent system because ACL has been not subject to this thesis. On the other hand, there are research topics covered by the JAE project which are beyond the scope of FIPA and are missing in the standard:

- Security with authority and certificates to authenticate users and agents has been subject to JAE. Permissions, policies and delegations of permissions have been dealt with in the JAE project.
- The agent transport protocol has been improved supporting persistence, restorability, and fault-tolerance.

During the JAE project the problems with disconnected mobile devices and mobile users have been a recurring issue of migrating mobile agents. Therefore, the mobility support on the application level has been studied intensively and become an important research topic which is completely missing in FIPA. With the support of security and mobility another requirement of

JAE has been user mobility that has been implemented in JAE and will be also discussed in detail in the next chapter.

3.6 Conclusion

In this chapter, the fundamental work of the thesis is described - the Java Agent Environment, JAE and its agent system. The overview of the JAE architecture is given by the Java based mobile agent middleware and the detailed description of the components of the agent system. The core technology for agent mobility, agent security, agent services and agent communication is discussed in detail with hierarchical module representation and class structure as well as process diagrams.

The service trading facilities introduced by the so-called *service center* with implementation details and programming examples to mediate the *service-based agent programming* paradigm is another focus of this chapter. Several measurements with different service trader approaches and varying number of requests to the service center emphasize the importance of the service center, introduced by the implementation of JAE. The combination of this new service trading and the integration of the Agent Directory by means of the *Agent Directory Protocol* (ADP) introduces the *plug-and-participate* concept of the JAE agent systems.

The agent migration is a key to mobile agents and therefore, much effort concerning security, persistency, and efficiency has been put into the research of the *Agent Transport Protocol* (ATP). The consideration of mobile devices and thus, the support of mobile computing has lead to the Ticket concept introducing ATP manipulation, security contemplation, and mobile agents' resource requirements. Examination of implementation details shows the interplay of the ATP, the persistence storage, and the protocol handler, clarifying this complex subject. Practical tests and measurements underline protocol security and code persistency, but at the same time, make clear the loss of performance in the agent migration.

The blackboard communication, based on the Linda architecture, and the message passing communication have been proven to be important and complementary concepts supported by the *Agent Communication Protocol* (ACP). Chapter three ends with a brief description of these two communication concepts and the outlook on JAE and FIPA.

Chapter 4

Mobility Support

The user mobility and the wireless data communication is a major topic in the *Universal Mobile Telecommunication System* (UMTS, [UMTS]) within the *European Telecommunications Standards Institute* (ETSI, [ETSI]) and the *International Mobile Telecommunication 2000* (IMT-2000) within the *International Telecommunication Union* (ITU, [ITU]). The explosive development of wireless access networks towards IMT-2000/UMTS and the ever-increasing popularity of the Internet have led to a growing interest in merging the access services to telecommunication networks and the Internet. Both standards aim to support a wide range of voice and non-voice services, narrow-band and wide-band, focusing on data communication services based on IP technology (packed-switched data, [RFC791]). They will give the mobile user similar performance as in the fixed network and allow the usage of new distributed and novel multimedia applications. The vision of any information, any time, anywhere becomes reality. More and more mobile phones flood the consumer market with integrated *Personal Digital Assistants* (PDA) and Internet access. Mobile users equipped with notebooks, PDAs, and mobile smart phones already have the freedom of terminal mobility and are able to access the Internet services. In addition to this amenity, user mobility is likewise becoming important to ensure the complete independence of users from the terminal or the mobile device they are working on. This leads to the need for more intelligence on the network side [IPCM97, IPCM98].

Today, the applications for mobile devices and the services offered by providers are mainly based on client/server technology, which is already used successfully in the Internet, e.g. optimized World Wide Web access by the *Wireless Application Protocol* (WAP, [WAP02]), email, news, etc. In the foreseeable future, mobile computing and user mobility will profit additionally from agent technology, which perfectly meets the requirements of nomadic computing with its capability to operate disconnected and without further interactions with the originating device or user. Mobility and autonomy allow mobile agents to move from their point of origin into a network and continue to operate even if the originating device is temporarily or perma-

nently disconnected from the network. This technology has the potential to provide mobile users with a higher degree of flexibility and efficiency by performing much of the work on the agent systems in the fixed network. The mobile agent approach helps to reduce required network resources and supports systems that do not have permanent network connections offering new value added services. Standardisation bodies have defined the management of mobile agents in general, but have not addressed the administration of mobile agents within a wireless access network with both nomadic users and mobile devices. Mobile agents act on behalf of their mobile users through simple transactions via a wireless access network. Agent systems on the fixed network side deal with the tasks to be performed, including for example looking for the required services, and evaluating returning agents.

Problems arise when returning or malfunctioning agents cannot be delivered immediately to the mobile device, due to non-reachability of switched off mobile devices. Here, it becomes crucial to prevent disappearing agents. Although packet-based bearer services like *Short Message Service* (SMS, [SMS02]) and *General Packet Radio Service* (GPRS, [GPRS01, GPRS02, GPRS03]) may guarantee a delivery, let us consider thousands of mobile agents, which have to be stored at the wireless message gateway. How much buffer are the bearer providers willing to offer each user, when the mobile device is switched off for more than one day? Therefore, alternative mechanisms handling returning and malfunctioning agents in the fixed agent network are required. Founded on the basic idea of JAE, service agents are proposed and provided to guarantee a reliable, secure, and scalable management of mobile agents at the border of fixed and wireless networks.

In the first section, a brief introduction into the cellular and the wireless local area networks is given. Subsequently, the architecture of the *Narrow Band Socket* (NBS, [NBS97]), proposed by Intel and Nokia is discussed, which allows a transparent data communication over narrowband bearers like SMS or GPRS for applications. On the basis of the *Global System for Mobile Communication* (GSM, consult the GSM Association for general information [GSMA]) and the NBS architecture, the problems with mobile agents and the requirements of the agent network and the agent system at the border to wireless access networks are described. An obvious solution to management of disconnected and malfunctioning mobile agents is given by the so-called *kindergarten* and *maintenance* concept which are both realized with the service-based agent programming paradigm of JAE.

The second objective of this chapter is the description of user mobility and the support of personalized service access with mobile agents. The problems to solve on the one hand are how to support the mobile customer regardless of the terminal employed and on the other hand, how to grant access to individually configured services. It is necessary to give users admission to their information in a *plug-and-participate* manner, keeping the details of the configuration and the underlying mechanisms transparent. Finally, persistency consideration in the agent system architecture and measurements of the implemented agent transfer protocol in the local network as well as in the wide area network conclude the chapter.

4.1 Wireless Access Networks

A fundamental component of mobile computing is undoubtedly the dynamical access to the fixed network. Besides mobility and remote computing, it implies dial-up with cellular phones or ad-hoc connection to wireless networks, notebooks, PDAs, communications servers and, of course, the user on the move. The ideal case is reached with instant communication and information exchange, any time and anywhere by means of wireless or wired media of different characteristics from changing locations (local or remote), and using communication or computing devices of different types and capabilities. Over the last decade, wireless communication technology has experienced phenomenal development, beginning early 1970 with the simple first-generation analog technology for business use, until the second-generation digital wireless telecommunications systems for residential and business environments (mid 1990), which have lasted until today. In Europe and in many other countries outside Europe, GSM has been established and currently, the 3rd generation (3G) of the mobile telecommunication system is launched. Figure 4-1 shows the coverage capabilities of the mobile telecommunications systems since the beginning of mobile communication [Rap95]. Digital mobile speech has already reached a high quality and the focus in *Personal Communications Services* (PCS) concerns high bit-rate data communication and global availability. Thus, the complete PCS will efficiently enable transfer on any form of information between any desired locations.

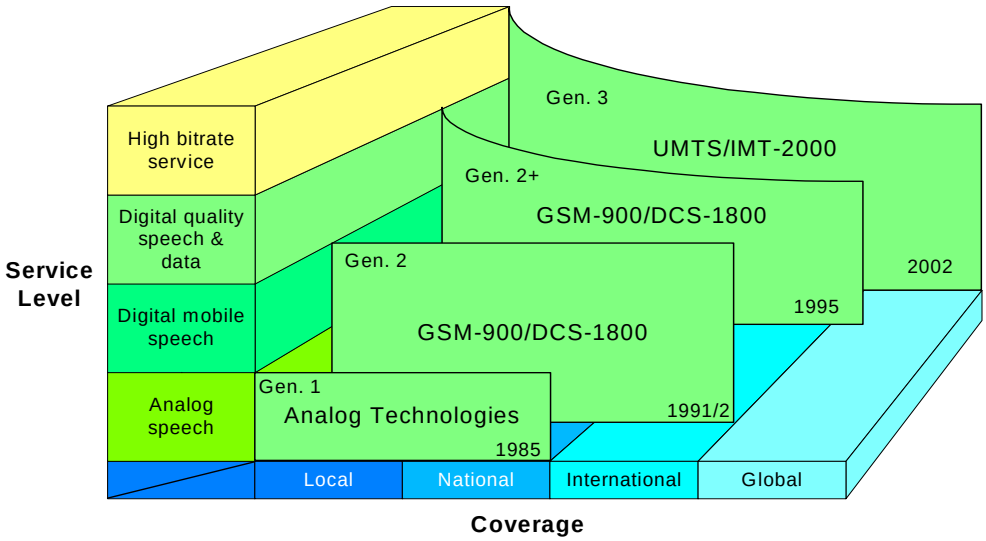

Figure 4-1: Service Level and Coverage Capabilities of Mobile Telecommunication System Generations

The 3rd generation of mobile telecommunication systems has been defined and pointed out to be very difficult due to the diverse view points of carriers and manufacturers in different countries. Although a compromise for the IMT-2000 has been found on the access technology, *Code-Division Multiple Access* 2000 (CDMA, [DGNS98, OjPr98]), European companies and research institutes are not forced to freeze the development of their own standard W(ideband)-CDMA/UMTS [UM3004] and GSM-900/DCS-1800 (*Digital Cellular System*, [DCS96]). In

Europe, tremendous effort has been spent to promote the development of the 3rd Generation Mobile Telecommunications Systems. The ACTS (*Advanced Communications Technologies and Services*) program has funded almost 30 research and development projects for the evolution towards UMTS in the period of 1995 until 1998. These projects have provided opportunities to master and trial mobile and personal communications services and technologies, involving service providers, communications operators, and equipment manufacturers. An overview of these ACTS projects are given by [ICM98]. Own experiences within the ACTS project *OnTheMove* [Har96, OTM98] have lead to this work after it was recognized that integration of mobile applications, services, and terminals with the fixed network, realized by conventional client/server-technology, has one major drawback in the scalability of the server and disconnected operations. The solution proposed by the OnTheMove project will be described and compared to the agent-based approach in Chapter 4.2.

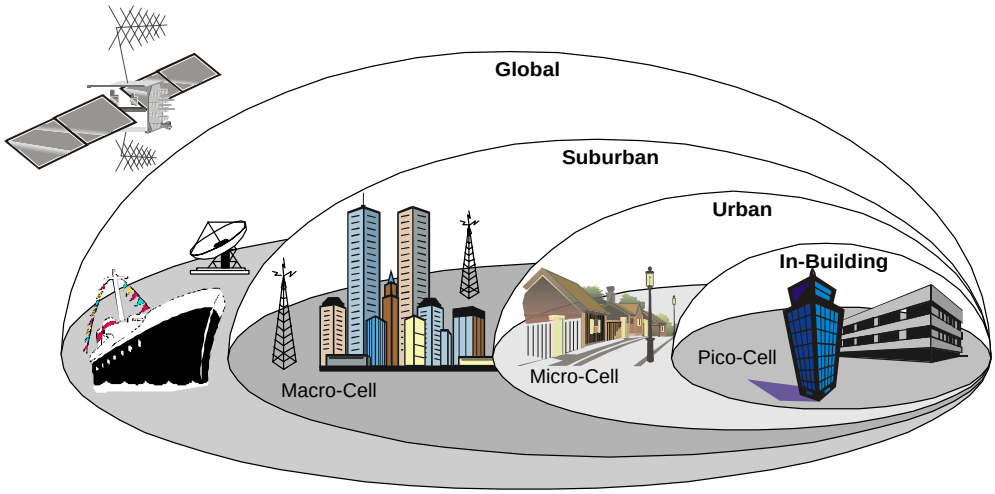

Figure 4-2: Environments of the 3rd Generation of Mobile Telecommunication Systems

Certainly, the above mentioned reasons have influenced the further improvement of the GSM network with the packet-switched network, GPRS, and the high-speed transfer technology dubbed *Enhanced Data Rates for GSM Evolution* (EDGE). EDGE have been subject to standardization in Telecommunication Industry Association (TIA) and ETSI Special Mobile Group (SMG). The process has been finalized at the end of 1999 and the technology is to achieve transfer rates up to 384 kbps in wide area networks, and 2 Mbps for local area coverage by a new channel coding technology in comparison to GPRS, foreseen for bit rates of nearly 170 kbps [FMMO99].

In spite of all the diverse but still common goals of Europe, America, and Asia, a unified *Personal Communications Network* (PCN) and PCS is envisaged. A more or less uniform view on the system environments is presented in figure 4-2. The seamless communication between the 4 zones, namely in-building, urban, suburban, and global, is one major objective of the 3rd generation PCN. In the case of UMTS, it is conceived as a multi-function, multi-service, and

multi-application digital mobile system that will provide PCS at rates ranging from 1.2 kbps up to 2 Mbps, depending on the specific environment and zone [IPCM95]. The announced high-speed transfer technology for GSM brings the present-day PCS closer to the next generation, fulfilling the throughput of most teleservices. The envisaged services, the proposed throughput, and the target bit error rates can be taken from the following table.

Teleservices	Throughput (kbps)	Bit Error Rate
Telephony	8-32	10^{-4}
Teleconference	32	10^{-3}
Voiceband data	2.4-64	10^{-6}
Program sound	128	10^{-6}
High quality audio	940	10^{-5}
Voice mail	32	10^{-3}
Video telephony	62-384	10^{-7}
Video conference	384-768	10^{-7}
Short messages/paging	1.2-64 (1.2-2.4 typical)	10^{-6}
Electronic mail	1.2-64	10^{-6}
Telefax (G4)	64	10^{-6}
Broadcast/multicast	1.2-9.6 (2.4 typical)	10^{-6}
Public voice announce	8-32	10^{-4}
Unrestricted digital data	64-1920	10^{-6}
Database access	2.4-768	10^{-6}
Teleshopping	2.4-768	$10^{-6}/10^{-7}$
Electronic newspaper	2.4-2000	10^{-6}
Remote control services	1.2-9.6	10^{-6}
Location and navigation	64	10^{-6}
Telewriting	32-64	10^{-6}

Table 4-3: Proposed PCS for the 3rd Generation System

The importance of an IP based infrastructure for the PCN has been taken into account by the leading European manufacturers and global leaders in wireless communications. They have recently set up the 3G-IP consortium with 9 well-known telecommunication companies[1]. All these improvements of GSM are indications for the smooth and progressive migration from the 2nd generation infrastructure to the 3rd generation of the mobile telecommunication system, instead of older design solutions, such as a stand-alone system or the integration with a fixed network. Furthermore, it is a clear sign that packet-switched networks, in particular IP based core networks, prevail and will allow a more transparent and efficient crossing to the Internet. This is a good forecast and motivation enough for the mobile agent systems to support mobile computing, because problems on application level still remain. Having the IP support of the next PCN in mind, the following discussions will focus on *Wireless Local Area Network*

1. Ericsson, Lucent, Nokia, Nortel, AT&T, BT, Rogers Cantel, Telenor, Telecom Italia Mobile

(WLAN), GSM, GPRS, SMS, and NBS being the current bearer services and underlying system for the access to the fixed network.

Wireless Local Area Networks [WLAN99] are important components for the in-building systems of the next generation PCN. Based on straightforward bridge architecture, WLAN provides transparent connection to a wired LAN through multiple access points (see figure 4-4). It allows operation in a full cellular network, supporting seamless, instantaneous roaming at a data rate of up to 2-11 Mbps, which is sufficient for the above mentioned services. Latest WLANs support data rates with more than 54 Mbps. TCP can be transparently set-up upon the physically distributed system, providing a simple connection within the local area network, which has been validated by own measurements varying packet sizes versus the signal to noise ratio at the air interface of *WaveLAN*[1], as shown in figure 4-5. The irregular throughput of the data is lead back to the windows advertisements in TCP [Lud00]. Worth mentioning is that the signal to noise ratio, which has nearly no effects on the data throughput, the used modulation technique *Direct Sequence Spread Spectrum* (DSSS), and the *Medium Access Protocol* (MAC), *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA), causes only low error rates (declared to be better than 10^{-8}).

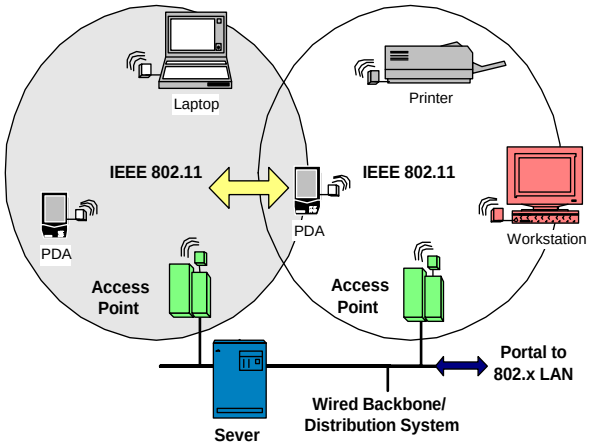

Figure 4-4: IEEE 802.11 Architecture

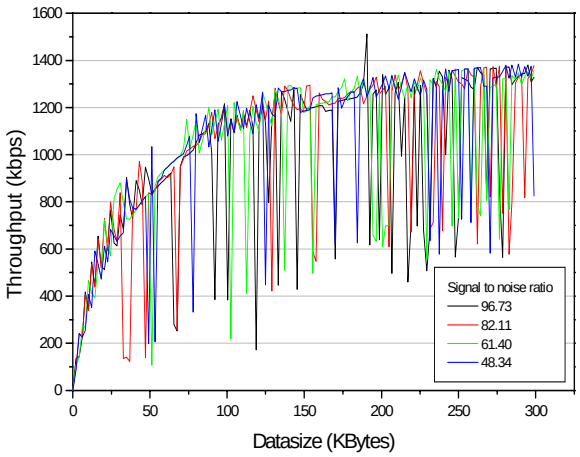

Figure 4-5: WLAN 802.11 TCP Throughput Measurements

Although WLAN enables terminal mobility within the networks coverage of the same IP subnet, it is a major objective of future mobile networks to support global terminal mobility with seamless roaming between different access networks such as GSM, GPRS, or DECT (*Digital Enhanced Cordless Telephony* [DECT03]). DECT has been specified by ETSI as the mandatory standard for digital cordless systems for a wide range of high-density wireless applications, both private and public. Thus, today it is a widely accepted standard for in-building telephone systems. The in-house data communication with DECT is also of interest with a net

1. WaveLAN is a product of Lucent Technology

maximum sustainable full bi-directional throughput (per transceiver) of 1 Mbps up to the sustainable unidirectional possible throughput of 2 Mbps up- or downlink [DPRS03]. In view of the environments of figure 4-2, the most eligible system in case of the transition from the in-building to the urban and suburban zone is GSM. Although, focusing on data communication, it must be taken into account that from the beginning, GSM was not designed to be the bearer service for high data communication. The basic operation, started in 1991, consisted of telephone services and some appropriated supplementary services, which are complementing the control and modification and are only usable in connection with telecommunication services.

The GSM standard differentiates 3 sub-networks, the *Radio-Subsystem* (RSS), the *Network-Subsystem* (NSS), and the *Operation and Maintenance Subsystem* (OMS). Components of RSS are the *Base Transceiver Station* (BTS), and the *Base Station Controller* (BSC), which control the transmitting power and the roaming between the radio cells. The BSC controls several BTSs and covers, such as the radio link switching system, the radio engineering aspects of a *Base Station Subsystem* (BSS). The establishment of connections, control of telecommunication services, and the through-connection of GSM from or to the fixed network, such as the *Public Switched Telephone Network* (PSTN) or the *Integrated Services Digital Network* (ISDN) are realized within the NSS. The *Mobile Switching Center* (MSC) is the central component of it, which exchanges information with the different registers and other MSCs by means of the Signalling System No. 7 (SS7) of the ITU. This means that there are more than one MSC which again can control more than one BSC. The connection to national fixed telecommunication networks are established by the *Gateway MSC* (GMSC), whereas the *International Switching Center* (ISC) set up connections to the international *Public Land Mobile Networks* (PLMNs).

The management of individual subscriber's information belongs to the key tasks of the NSS, i.e. registration, authentication, location, hand over, and control of routing information. Essential for connection setup and service supply is the *Home Location Register* (HLR), a central database (but not necessarily the only one depending on the number of subscribers and the organisation of the network), that contains all information of a subscriber who belongs to this "home" area in the long-term. Accordingly, the *Visited Location Register* (VLR) is a local database that contains a subset of the subscriber information required for a call setup in a remote radio cell. The update of the VLR is accomplished by the roaming mobile station and the HLR. For a secure authentication of both the mobile user and the mobile station, the *Authentication Center* (AC) and the *Equipment Identity Register* (EIR) are located in the Operation and Maintenance Subsystem. With the *Subscriber Identification Module* (SIM), each user can be unequivocally identified and localized. Correspondingly, a mobile station and its right of use can be defined and checked with the *International Mobile Equipment Identity* (IMEI). Servicing and operation are also controlled by components in the OMS. For a central remote control of the BSS, both the *Local Maintenance Terminal* (LMT) and the *Operation and Maintenance Center Radio* (OMC-R) are designated to it. The *Operation and Mainte-*

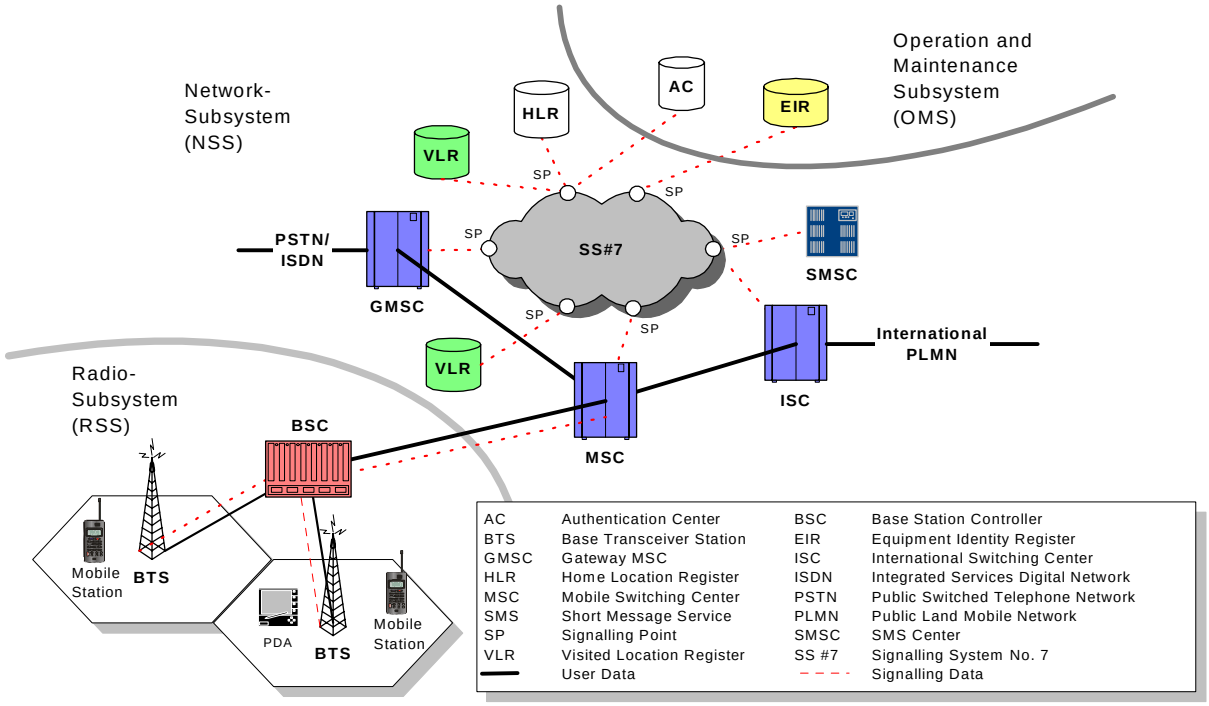

Figure 4-6: Architecture of the GSM Mobile Switching Network

nance Center Switch (OMC-S) is linked to the MSCs as well as the registers and takes care of the NSS. The overall architecture of GSM is outlined in figure 4-6, but does not show the LMT, the OMC-R, or the OMC-S. For full details on GSM, one should consult simultaneously the ETSI standards in e.g. [EbVö99].

Having introduced the architecture, the remaining question in this context is how data communication is supported with GSM. The bearer services offer asynchronous or synchronous data transport capabilities with circuit-switched or packet-switched data rates of 300 to 9600 bps, and a 13 kbps bearer service for voice. Unfortunately, the maximum data throughput of GSM phase 2 is not sufficient for most of the future teleservices (see Table 4-3). In the transparent mode, a circuit-switched connection between the mobile station and the interworking module of the MSC ensures a constant bit rate, constant transport delay, and a residual bit error ratio with the use of the *Forward Error Correction* (FEC), but is certainly dependent on the current channel conditions. A more reliable connection is given with the non-transparent mode that additionally activates the *Automatic Repeat Request* (ARQ) process and the special layer 2 *Radio Link Protocol* (RLP), which is adapted to the GSM radio channel. With the ARQ, the retransmission of blocks is initiated when residual errors could not be corrected by the FEC, which results in a significant reduction of the residual error rate. However, with changing error behaviour of the radio channel, the frequency of repetitions increases and thus, the average transfer delay and the net bit rate of the data service decreases. Figure 4-7 shows repeated measurements based on GSM and IP/UDP with varying data throughput depending on the connection mode. The measurements show that on the one hand data throughput vary depending

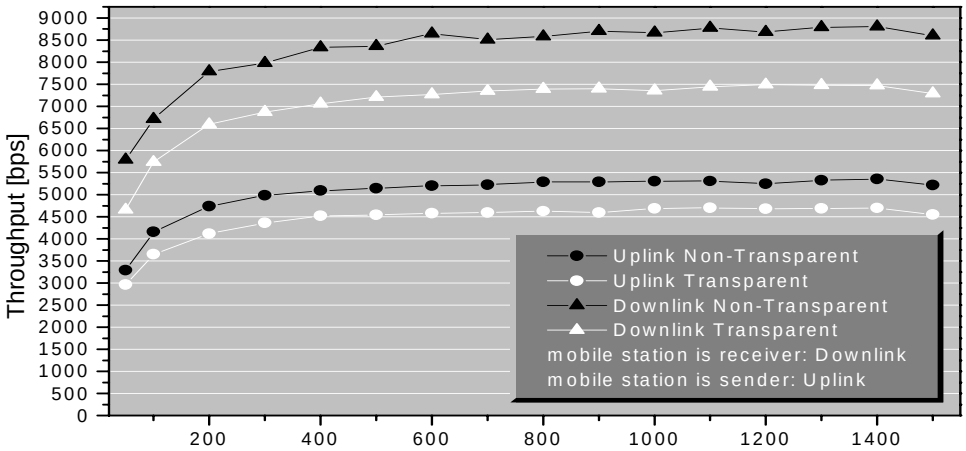

Figure 4-7: UDP Measurements in GSM

on the selected mode, and on the other hand, depending on the mobile station sending or receiving data. Not considered in the measurements are the proneness of the radio channel, which can be traced back to high fading rates, deep shadow holes, limited number of frequencies, hand-over, interference by means of radio wave reflections, or heterodyning of frequencies of different BTSs. Additional, intensive interferences can lead to a complete interruption of a connection, which is a problem that has to be considered in mobile aware applications.

Referring to the data communication, the teleservice SMS has an even worse throughput due to the fact that it is a point-to-point message service to and from GSM mobiles with the limited size of 140 octets per message. The exploitation of the signalling channels requires the size limitation of the messages, but on the other hand guarantees continuous availability, even during calls. Furthermore, the inclusion of the *SMS Center* (SMSC), which acts as a store and forward center for short messages, vouches for the delivery to a mobile station. The SMS standard distinguishes between mobile originated and mobile terminated messages. Mobile originated messages are sent from a mobile station to the SMSC, whereas mobile terminated messages are sent from the SMSC to a mobile station. This is of interest for transferring not only messages by other mobile stations but by a variety of other sources, e.g. telex, facsimile, or message handling systems according to ITU X.400 [X.400]. Hence, SMS becomes interesting for IP based networks to serve as the bearer service in wireless access networks. A major advantage of both the SMS and the much more promising GPRS is that an awkward dial-in is dropped and the radio channel can be better abused in comparison to the circuit-switched connection.

GPRS is especially interesting for diverse applications when the mobile data communication is typically characterized by bursty traffic, e.g. in logistics application for coordination of trucks. Thus, the essential difference is the link-by-link transmission between the mobile station and the BTS via a logic channel, instead of the end-to-end transmission for the duration of the connection. Moreover, the packet-switched data service will allow charging depending on the

amount of data transmitted and the quality of service negotiation. Seen from the user's point of view, it is just another IP sub-network and data packets are routed by means of *GPRS Support Nodes* (GSNs) between different Intra-PLMN IP backbones and general IP data networks (Internet, Intranet, or *Internet Service Provider* (ISP)). Depending on the *Packet Data Network* (PDN) interworking model, a transparent and a non transparent access to the external IP data network has been defined. The transparent mode is applied in the case of direct access to the Internet involving the *Gateway GSN* (GGSN). The mobile station is given an addressing either dynamically at protocol context activation or statically at subscription, both belonging to the operator's addressing space. This address is used for packet forwarding between the Internet and the GGSN and within the GGSNs. The authentication/authorization process is not performed during this protocol setup.

The non transparent mode is foreseen for Intranet and ISP access, which requires an address allocation server belonging to the Intranet/ISP addressing space. In this case, the GGSN serves as the logical interface to the Intranet and ISP, respectively, and controls the dynamic and static addressing requests as well as the authentication with the support of the *Serving GSN* (SGSN). The GGSN obtains the protocol configuration options from a server belonging to the Intranet/ ISP. It maintains routing information to tunnel the *Packet Data Units* (PDUs) from the SGSN to the external networks, and vice versa to the mobile stations when the authentication has been successful. For detailed information relating to the GPRS network, the standards are recommended (e.g. [GPRS03]). An example of an architecture with both a narrowband network on the upper left side and the GPRS network with transparent access on the upper right side is shown in figure 4-8. The diagram is split in two parts, showing the architecture in the upper half and the protocol stack in the lower half.

In 1997, Intel and Nokia have considered bearer services like SMS and Mobitex[1] as an attractive and cost-effective wireless narrowband network. They have proposed and implemented the *Narrow Band Socket* (NBS) to realize a transparent link over the wireless access network to the Internet. Applications on the mobile station are to be used with only little changes in the socket code part. The specifically adapted NBS takes advantage of the property of the narrowband network and realizes an efficient PDU transport over the wireless access network [NBS97]. Today it is most commonly used with a specific port number, known as the Narrow Band Socket port number, in the header of the Smart Message [SM00]. The originally designed Narrow Band Socket protocol is no longer used and the WAP Forum's *Wireless Datagram Protocol* (WDP, [WDP00]) is recommended to be used instead when supported by the target device.

So far, the architecture of selected wireless access networks, in particular Packet Data Networks, have been described to give an understanding of the requirements for IP. In the next section, the support of mobile computing in IP-based networks will be described and solutions

1. Mobitex is a packet switched system for mobile data communication developed in Sweden.

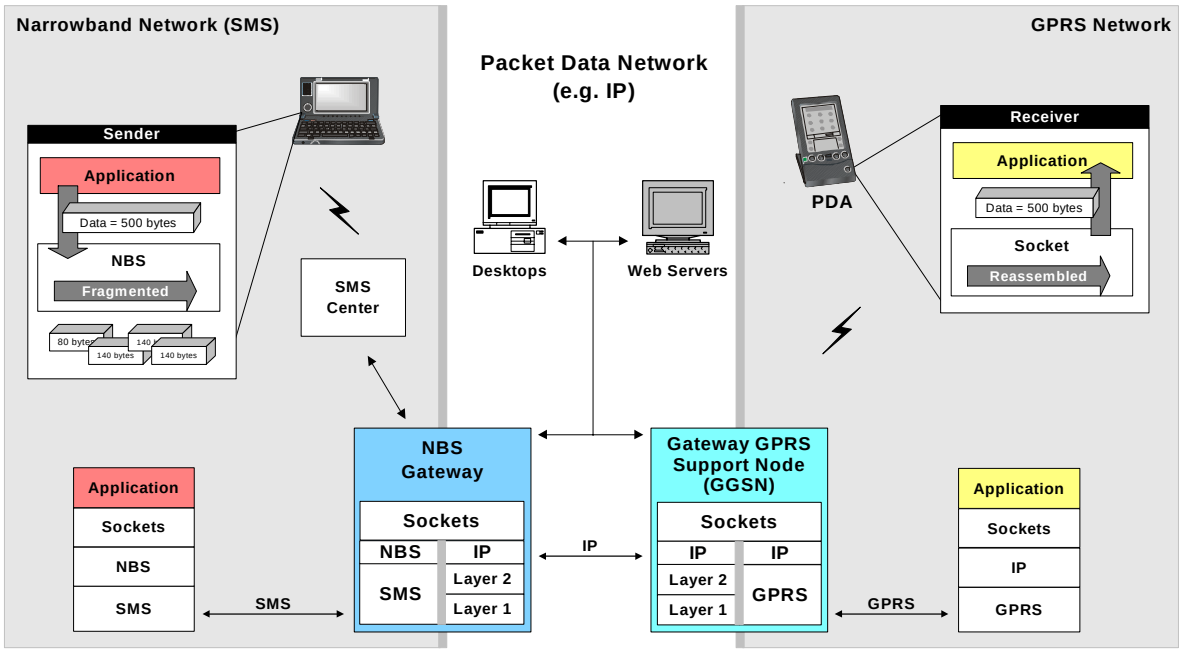

Figure 4-8: Example of an IP Backbone coupled to a Narrowband Network and GPRS Network

are proposed with the conventional client/server technology and with the mobile agent system. The common problem with application protocols on top of TCP will be the non-reachability of switched off mobile stations or released devices from the network. This is the subject of the next section - and the autonomy of mobile agents promise an elegant, but simple solution.

4.2 Support of Mobile Computing

The first association that comes with mobile computing is terminal mobility, which is also the primary goal of the wireless access networks. That is more or less realized with the above mentioned architectures. WLAN, for example, ensures terminal mobility within the networks' coverage of the same IP subnet. The same applies to data communication with DECT, but the major objective of future mobile networks is to support both global terminal and personal mobility with seamless roaming between different access networks such as WLAN, DECT, GSM, GPRS, and so forth. On the one hand, there is the problem of how to support temporarily connected mobile devices with changing IP addresses. On the other hand, how is personal mobility supported? - because it is strongly related to the field of application. With personal mobility, it must be possible to identify users regardless of their current location. The users must be able to address and receive connections from any terminal, whether it is fixed or mobile, a PDA or laptop. This involves that the mobile user may profit from his personal service profile and that she is reachable at the same address, independent of the terminal she is currently registered with and to which sub-network she is currently attached.

4.2.1 Mobile IP

The *Mobile IP* [RFC2002] Task Force works on new routing processes for integration of mobile devices to TCP/IP based networks. Mobile IP is a modification to IP and allows nodes to receive datagrams anywhere they happen to be attached. Therefore, a *home agent* is introduced, which acts as the re-addressing node on the home network using the *Location Directory* (LD). The LD keeps the location information of the mobile devices, and the home agent forwards all datagrams that arrive on the home network via standard IP routing, to the current location of the devices. The current location is represented by a *foreign agent* in the remote network that updates the LD of the attached mobile devices and at the same time, delivers the datagrams to the mobile nodes. All datagrams sent by the mobile device are delivered to their destinations by the standard IP routing without any re-addressing [Per98]. Although this triangle communication has drastic loss in performance [Fas98], it is a proposed standard and a solution for the *macro mobility management* that keeps communication protocols underneath TCP transparent. The simplified routing example of Mobile IP is illustrated on top of the GPRS architecture in figure 4-9.

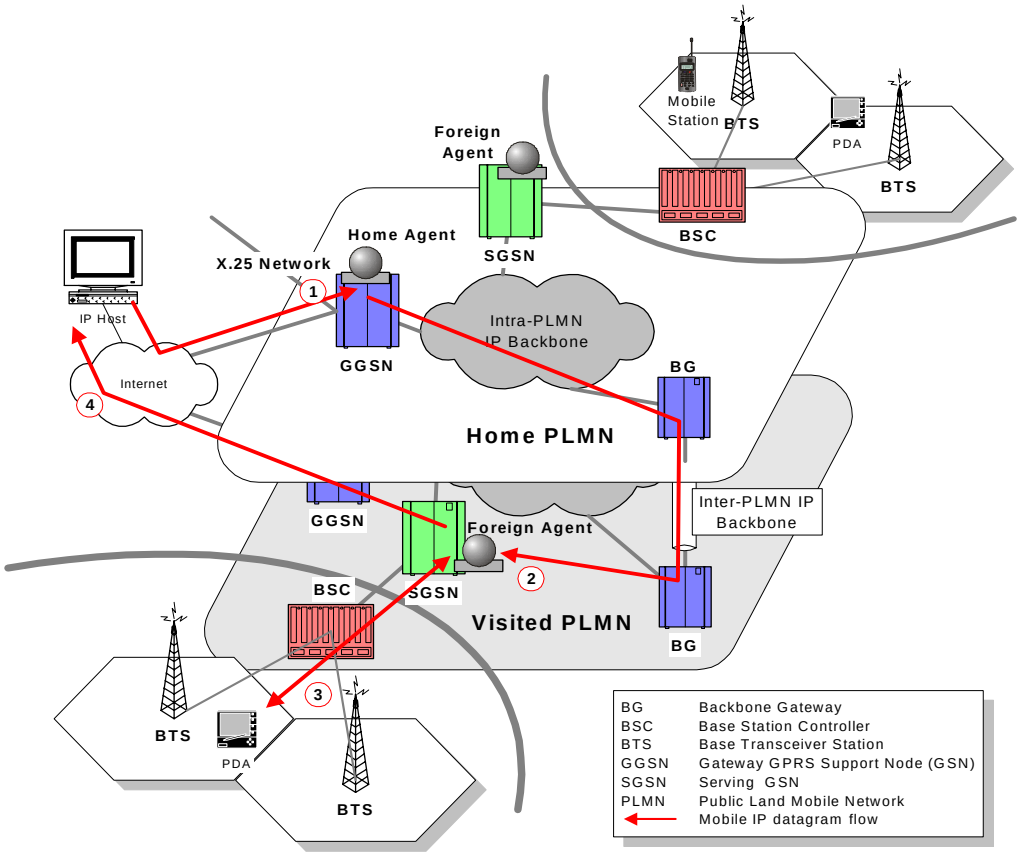

Figure 4-9: Mobile IP Triangle Routing above the GPRS Architecture

IP based data communication always requires a unique IP-address that, in most cases, is automatically assigned to the mobile device during the authentication to an Internet/Intranet subnet

or any Internet Service Provider. The *Dynamic Host Configuration Protocol* (DHCP, [RFC2131, RFC2132]) is supposed to simplify network attachment, providing all the necessary configuration that an Internet citizen requires. DHCP is recommended for mobile computing and can be combined with the foreign agent of the Mobile IP approach. The nomadic client may obtain necessary information such as the IP address, Domain Name System (DNS, [RFC1034, RFC1035]) server address, domain names, subnet prefixes, default routers, the X.500 server address, and many other similar useful information. All these proposed and standardized IETF protocols lead to a zero-administration concept in IP based networks.

Within the architecture of GPRS, the Mobile IP concept combined with DHCP could gain an interesting role - although Mobile IP does not meet the requirements of GPRS exactly - because non-Internet data packet protocols like X.25 are not supported. In figure 4-9, an example configuration is shown with the home agent located in the GGSN and the foreign agents located in the SGSNs. In the first step, any IP host sends datagrams to the Mobile IP address that by means of the home agent forwards (as the second step) the packets to the current hosting foreign agent. The information exchange at the border of the access network always takes place with the foreign agent on the SGSN. Returning datagrams are sent back directly from the foreign agent to the IP host without re-routing the packets to the home agent, as illustrated with the forth step in the figure. First approaches with roaming between different bearer services and the implementation of Mobile IP have been made in the ACTS project, *OnTheMove*. The prototype implementation of the mobile middleware and its mobility support is outlined as deputy for client/server technology in the next section.

4.2.2 The *OnTheMove* Mobile Middleware

The ACTS project, *OnTheMove*, has aimed at developing a standardized *Mobile Application Programming Interface* (Mobile API) to facilitate and promote the development of a wide spectrum of mobile-aware applications. In this context, a mobile middleware called *Mobile Application Support Environment* (MASE) has been developed, which is to support the variety of mobile applications working on future heterogeneous mobile communication environments by providing a common view to the underlying platforms. MASE addresses the functionality below the application and above the operating system. Hence, it allows applications to be platform independent and enables a smooth transition from current wireless networks (like GSM and DECT) to future UMTS networks.

Instead of directly accessing the operating system, mobile-aware applications make use of the Mobile API and benefit from simple access to MASE services, which hide the complexity of heterogeneous networks and operating systems from the applications. Thus, MASE simplifies the development of mobile-aware applications and frees them from the complex processing caused by additional needs when accessing heterogeneous networks and running on mobile devices [PaMe97].

MASE is distributed and runs on both, depending on hardware requirements, as a “light-weight” version on the mobile device and as a full version on the so-called *Mobility Gateway* (see figure 4-10). The latter acts as a mediator between the wireless and the fixed network

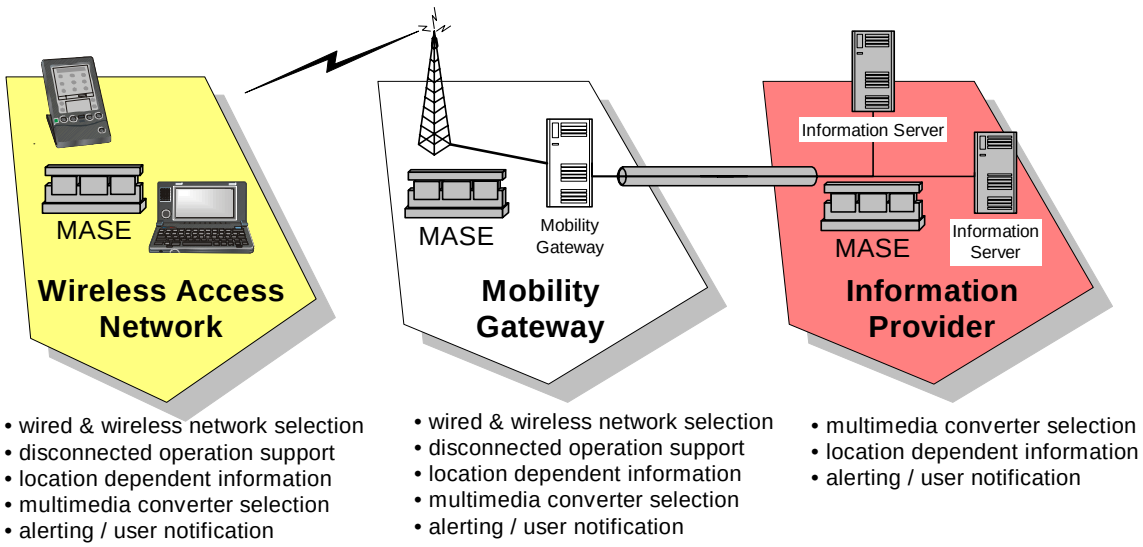

Figure 4-10: Distribution of the OnTheMove Mobile Middleware MASE

infrastructure. It works as a mediator for mobile clients which are typically connected over wireless access networks with lower bandwidth. MASE enables access to the *UMTS Adaptation Layer* (UAL), which provides applications and middleware components unified access to different underlying networks. The UAL hides network specific details by selecting the appropriate bearer service and protocol stack according to the requested QoS. Thus, MASE applications are aware of the current network Quality Of Service (QoS) through monitoring and management facilities of the UAL. Details of the UAL have been published and are described in [MPL96, PMT+97]. The general functional features of the UAL are:

- selecting appropriate transport protocols and configuring them for efficient use (e.g. adjusting packet size and timers to the bearer service parameters),
- selecting and configuring an appropriate bearer service,
- managing the roaming between different networks and bearer services as well as the bearer service switching.

The *General Support Layer* provides the functionality required for distributed systems. It implements object storing and caching as well as event handling and security services. On top of both layers, several so-called manager components are installed, providing different dedicated services:

- The *Location Manager* helps users to navigate in new environments. It deals with the position determination of the mobile clients applying GSM, wireless LAN and DECT cell finder capabilities and usage of the *Global Positioning System* (GPS). The abstrac-

tion is fully provided by MASE as the geographical information is accessible through a simple and uniform API and is independent of the mechanism used by the Location Manager to gather the location data.

- The *communication manager* of the MASE architecture consists of several components supporting *Hypertext Transport Protocol* (HTTP), alerting, massaging and multimedia e-mail communication. For example, the role of the HTTP proxy system is to improve the performance and usability of HTTP access over low bandwidth network connections (cellular, wireless LAN, modem). The requested multimedia objects can be processed/converted by the MASE to better match the actual network bandwidth, terminal characteristics (display capability, etc.) and user quality requirements.
- The *System Adaptability Manager* is responsible for the provision of optimized and personalized mobile services. User-, network- and terminal-specific QoS parameters are managed in profiles handled by the *Profile Manager*. These profiles are used to compute the best adaptation (by means of multimedia conversion) of the MASE communication services, taking into account the current network and end system QoS parameters as well as to the user's personal preferences.

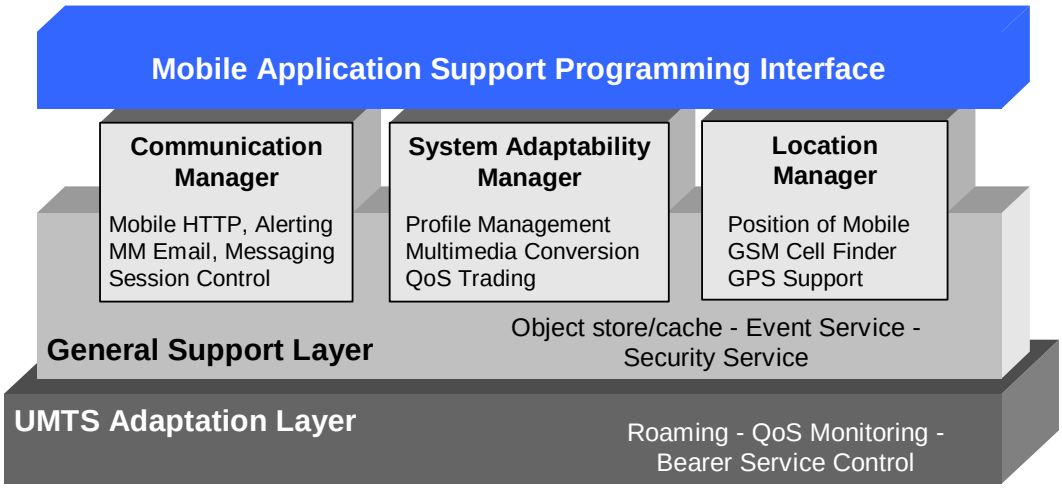

Figure 4-11: Overall Architecture of MASE

Figure 4-11 shows the overall MASE architecture with its corresponding building blocks. All the components shown have been implemented and tested in various testbeds. Detailed discussions about the testbed and the results can be gained, for example, from [KPM+98] and [FRG+99].

Considering the architecture of the mobile middleware, it becomes evident that both the mobile agent and the client/server technology have to coexist to fully support user mobility. While client/server technology is indispensable for streaming protocols, e.g. video applications (see [MSP97] for the usage of MASE in video applications) or immediate information gathering with direct interactions with the user [MPF+98], mobile agents seems to be better suited to

operate disconnected, which will be discussed in the next section. Finally, it can be stated that both technologies are of interest for the future development and that the UAL plays a significant role in keeping protocol stacks underneath the application protocol transparent to the software developers.

4.2.3 Middleware based on Mobile Agents

Since the beginning of the mobile agent technology, the asynchronous work of them has been considered the main advantage, particularly in the context of partially connected computers. From the beginning, the first commercial product Telecript have presented the agent technology for electronic commerce applications and small personal communicators. Although Telescript and Sony's Magic Cap - the PDA for Telecript agents and applications - have been a failure (among other things due to its proprietary), they continue still today to be a very good example of a language and framework for creating mobile agents. Today, the requirements for disconnected operations have risen. The support of mobile computing by means of mobile agent technology has to be more effective. It is necessary that the agent network is scalable, the whole environment is transparent to the user as well as to the applications, and last but not least, agent systems operate in a plug-and-participate manner. Thus, several requirements must be fulfilled by the agent system additionally (compare to the list in Chapter 3).

- mobile agents must be able to migrate from a partially connected computer (such as a PDA or notebook)
- mobile agents must be able to work autonomously without any further user interaction while in the disconnected operation mode
- mobile agents must be able to return (at least the results) to the owner independent of the mobile device (user mobility)
- faulty mobile agents must be captured, stored, and handed over to the user
- mobile agents must incorporate customer profiles
- wireless networks have to be considered in the transport protocols
- agent systems on any devices must be able to transparently plug to any agent network and participate

Many agent systems claim to support disconnected operations, but most of them only provide a very simple feature such as Telescript's *goHome* method. These systems do not cater for the increasing number of potentially moving agents, and the demand for user mobility is largely ignored. Mobile agents that cannot reach a destination need to wait for its reconnection on an agent system, despite the fact that the user may already be reconnected to the network through another machine. A similar solution is discussed in a technical IBM report [CGH+95]. In this scenario, a mobile agent has to wait on a predetermined agent system (called the *Agent Meeting Point* (AMP)) until the mobile client is reconnected. Other agent systems such as Mitsub-

ishi's Concordia, IBM Aglets, or IKV++ Grasshopper, to name only the commercial ones, offer similar solutions and do not further discuss this topic.

More intensive research about mobile agents for mobile computing has been done at the Dartmouth College with the Agent TCL, now renamed to D'Agents [GCK+02]. They have introduced a so-called *docking* system that allows an agent to transparently move between partially connected notebooks, PDAs, or modem-connected home computers. To overcome the problem of inaccessible mobile computers, the docking concept pairs reconnecting computers with a permanently connected *dock machine*. On the one hand, this dock machine hosts waiting mobile agents, and on the other hand, it keeps track of the changing access points of the mobile computers. Thus, a mobile agent that has failed to reach a mobile computer per default jumps to the dock machine and is suspended until the reconnection of its destination is announced to the dock machine [KGN+99]. At first glance, the docking system appears to be similar to the kindergarten service in certain ways. However, only terminal mobility is considered: neither the user mobility, scalability of the agent network, nor the transparency of the network are taken into account. In JAE, mobile agents have a more transparent view of the network and therefore, fixed addressing schemes for agent programmers are not applicable.

All mobile agent systems mentioned above have considered the handling of mobile agents in environments with nomadic users. The basic idea of JAE’s support of mobile computing is similar to that of D'Agents, but wireless access networks have been investigated in more detail. Furthermore, a more reliable and better adjusted agent transfer protocol has been designed. Finally, the integration of mobile agent management with the service based programming concept of JAE allows a fully transparent and distributed support of user mobility.

The subsequent scenario helps to understand the necessary components and services involved in JAE to support user mobility and personalized service access and will be referenced throughout this chapter: imagine an extremely mobile user, who wants to be informed directly if some banking transfers take place, and who picks up most of the information on the move. This user is equipped with a GSM/GPRS enabled PDA that she is sharing with some other colleagues. The PDA has telephone capabilities and is running an agent-based middleware. In general, this PDA is ready for reception, but can be off-line from time to time. Obviously, other colleagues working with the PDA should not receive her banking information when she works in the office, but she will receive any information at her desktop. In general, we assume that the user mobility is coupled with a reasonable and secure user authentication at terminals, which is necessary to guarantee that each mobile agent can be assigned a unique ID. This is very important when guaranteeing user mobility and working with sensitive personal data. No problems will occur as long as mobile agents that have fulfilled their task are able to return to the originating device, but what happens if some tasks fail or the final destination terminal is not reachable? How can information for a dedicated person be redirected to the current terminal of that user? These issues are addressed by the kindergarten and maintenance concept in the following section.

4.3 Mobility Support with JAE

In JAE, the distributed Agent Directory (AD) plays a significant role as the central deposit for the descriptions and locations of agent systems as well as mobile and service agents. As mentioned before, the LDAP server (SLAPD)/X.500 directory service is deployed for this purpose. Although the use of the X.500 service directory/SLAPD bears the disadvantage of not having a reactive database - which would be able to generate notifications in case of predefined events - it has the crucial advantages of being both a standard (X.500) and distributed. Thus, if for example some services cannot be found in the domain, the distributed directory service may help with a redirection to other agent directories. In other words, having one reference to an AD means having a gateway to all other agent systems in the agent environment. This allows for a direct support of the *plug-and-participate* concept. For the purpose of personalized mobile agents and failure handling, the agent directory has been enhanced. It also serves as a persistent storage of data, which is required as a fundamental service to base the kindergarten and the maintenance concept upon.

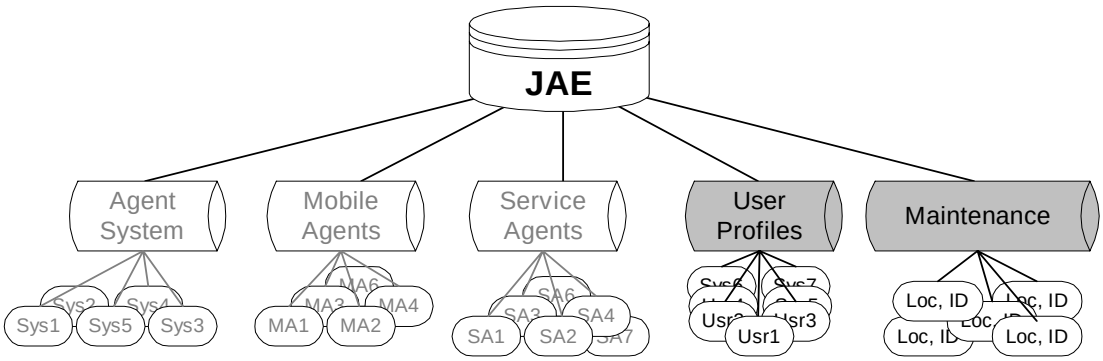

Figure 4-12: The Agent Directory Structure

So far, three main branches have been identified in the sub-domain of the agent directory JAE. All agent systems register during start-up with the agent system branch, and export service descriptions of automatically loaded system agents to the system agent branch, thus enabling all involved agent systems in this domain to locate and find services through a service look up. Likewise, all agent systems de-register both the agent system and the services offered at this system at a regular shutdown. In the same way, descriptions of mobile agents that intend to communicate or intend to be registered and tracked are stored in the mobile agent branch. As depicted in figure 4-12, the user profiles branch and the maintenance branch have been added.

Although the user profiles may be stored somewhere else, the maintenance branch is an enhancement of the agent directory structure and necessary as a basic feature of JAE to cope with faulty mobile agents and general failures of agent-based applications. In the prototype implementation, all user profile data are stored at the agent directory as well. Note that the user profiles need not necessarily be in the sub-domain of the agent directory. They contain impor-

tant information for individual configuration of mobile agents and services. The profile of a user should contain address information, email, telephone, and fax numbers as well as any other data of interest and should only be readable by authorized persons or applications. The profile data are encrypted and stored as an object. Only system agents or user applications on a trusted system should have access to this secure database. Mobile agents that are launched by a mobile device and merely have the ID and a public key of the user must first find a user profile service that provides them with access to the required user data. The combination of the user ID, the public key of the user, and the profile access service allows a mobile agent to have read access to the user information in the agent directory. This is a simple solution to store user profiles in the agent directory.

The decision not to implement profile data with system agents is due to the experience that system agents consume additional resources and the management of system agents cause further overhead. In general, personal data are not updated frequently, but are read for customized task performance of agents. Thus, to keep personal data also accessible from different applications and locations, e.g. with a web browser, the storage of the data in a directory service is reasonable. In the following section, the kindergarten and maintenance concept will be presented in detail, which utilizes the directory structure.

4.3.1 The Kindergarten Concept

The kindergarten concept is employed when a mobile agent explicitly tries to migrate to a destination host, which is currently unavailable. This can occur either on a service request (the system offering this particular service is unavailable, i.e. there is no connection or the system is down), or on returning to the user's device that is temporarily inaccessible, e.g. the PDA is switched off.

If the mobile agent intended to migrate to a destination system to use a specific service and it is not explicitly focused on a particular service agent, it can look up the availability of another identical service agent, which is located on another agent system. If there is such a service agent, the mobile agent can try to use this service agent, i.e. try to migrate to the corresponding agent system. Thus, simply changing the destination system solves the problem. However, if no alternative services are allowed and available respectively, or the mobile agent is not able to return to the user, a different solution is required.

In the following, the kindergarten concept and the modules involved will be described. These are the so-called *kindergarten services*, the *notify services*, and the central look up maintenance branch in the agent directory. Note that there can be more than one kindergarten service in the agent environment and also several notify services as depicted in figure 4-13.

A kindergarten service is realized through a system agent. Its main task is to suspend mobile agents for a specified time period and to register information about the agents and their desti-

nation with the maintenance branch of the agent directory, when the agents' requests cannot be handled. Accordingly, a notify service informs the kindergarten services when a previously inaccessible or a new agent system is coming up. For this, an agent system either executes the notify service during start-up, or a user explicitly starts the notify service in order to collect mobile agents she is waiting for.

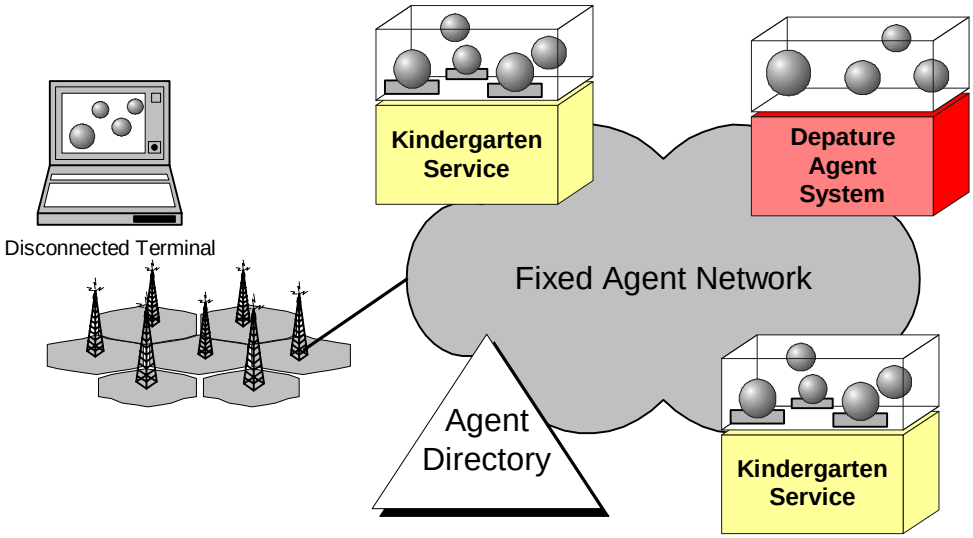

Figure 4-13: Distribution of Agent Systems and Services

In this scenario, a mobile agent is to inform the user about a bank transaction. Because a telephone call could be disruptive, a mobile agent containing the message is sent via GPRS. This mobile agent intends to migrate from its current location to the destination system, i.e. from the bank server to the user's PDA, which is temporarily inaccessible, e.g. because of bad radio coverage. If there is no other way of contacting the user, the mobile agent can either stop immediately or wait until the user can be reached. Neither of these two cases offer an acceptable solution. On the one hand, an agent can carry important information, possibly collected in a long and expensive chain of migrations which is lost if the agent simply stops. Waiting for a user or an agent system, on the other hand, might take an undefined period of time, possibly last forever. Hence, the immediate stop of the agent with the loss of its results, and busy waiting for the user are not reasonable. The kindergarten service solves this problem as follows.

If a destination system cannot be reached, an agent system will retry to reach the destination several times (number of retries specified in the *Agent Transfer Protocol* (ATP)). On failure, an exception handling will be initiated: the mobile agent will look up an appropriate kindergarten service and move to this service at a given agent system. At this new location, the mobile agent requests the kindergarten service for suspension until the destination system, i.e. a service or a user, is reachable or until a predetermined period of time has expired. It is the responsibility of the kindergarten service to wake up the suspended agent as soon as either of these two conditions is met. In our example scenario, a limited lifetime of e.g. one day makes sense,

because the bank transaction not interesting or reasonable after one day. If the predetermined time has expired, the agent will simply be deleted. The sequence diagram in figure 4-14 explains the notification mechanism if either an agent system is restarted or a user has been logged in to an agent system.

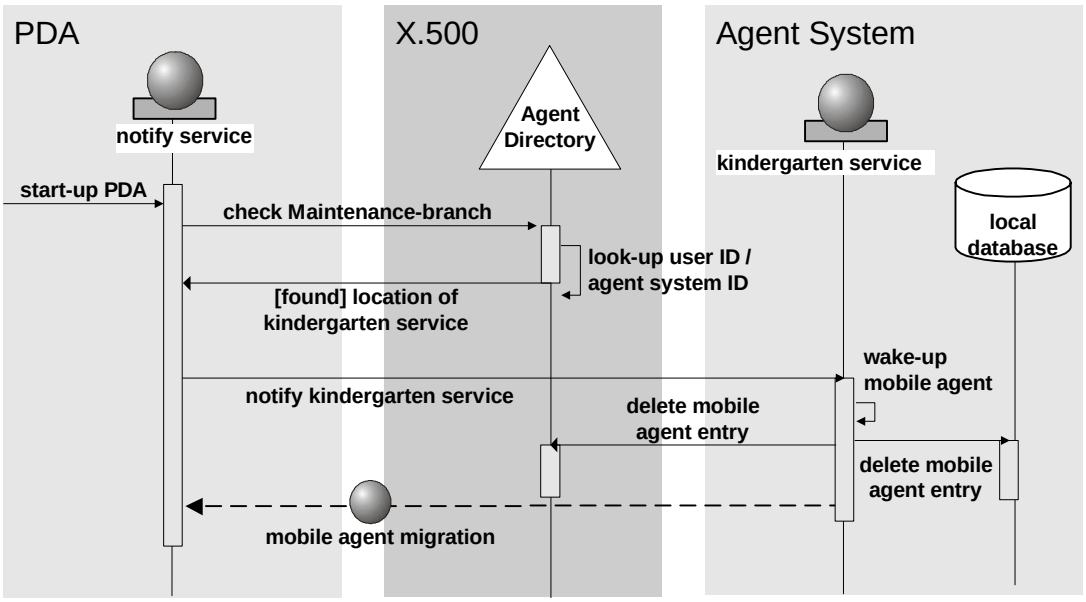

Figure 4-14: Sequence Diagram of the Kindergarten Concept

Depending on the mobile agent's destination, the kindergarten service stores information about the requested user or the destination system in addition to the mobile agent ID and the kindergarten service location. Thus, the agent directory entry in the maintenance branch consists of the following information: `([<user ID>|<agent system ID>], <mobile agent>, <location of kindergarten service>).` Once the mobile agent is suspended and registered with both the local database and with the agent directory, the following mechanism for invoking the mobile agent is used. Upon start-up, the destination agent system executes the notify service, which checks the agent directory entry in the maintenance branch for an agent system ID. If such an entry is found, the notify service receives the location of the kindergarten services with waiting mobile agents. The delivery of the notify message to these kindergarten services will activate the agents.

Returning mobile agents are re-started when the user manually executes a notify service or re-logs into an agent system. The procedure is identical to the one above, the look up in the maintenance branch, however, is done for the user ID. Having received a notification message, the kindergarten service checks its local database for the requested mobile agent. Upon re-activation, this agent obtains the location of the destination system and invokes the migration operation. Having notified the mobile agent, the kindergarten service deletes the corresponding entries from its local database and from the agent directory.

Taking a closer look at the deployment of the kindergarten service, one immanent problem of this concept becomes obvious. On restart of an agent system, there might be a large number of mobile agents, which have registered with a kindergarten service while the system was unavailable, and which now threaten to flood the restarting agent system. These mobile agents can then block the agent system so that no other execution is possible. This is particularly critical with agent systems running on PDAs.

According to the kindergarten concept, all of the kindergarten services notified by the notify service will wake up the corresponding mobile agents. To avoid the overload situation in the requesting agent system, the Agent Transfer Protocol (ATP), as described in the last chapter, ensures that each mobile agent is forced to send the *Ticket* to the destination system prior to migration. The destination system can check the available resources and send a reply to the mobile agent, indicating whether or not a migration is allowed. For example, to avoid overload situations at the destination device, a mobile agent is only accepted if the maximum number of active agents is not exceeded. The interplay of the kindergarten concept and the Agent Transfer Protocol can deal with the problem of uncontrolled flooding of small devices with mobile agents and guarantee an adequate handling.

If the agent migration or its execution in general can no longer be properly continued, a means for retrieving error information and possibly the agent's state must be given. For example, if the last handshake of the ATP fails, the delete command can be neither received by the sender of the mobile agent, nor can the acknowledgement of it be received by the receiver. At this point, the maintenance concept comes into play to prevent multiple copies of a mobile agent from being executed and to save results of the mobile agent.

4.3.2 The Maintenance Concept

The immanent problem of disconnected operations with mobile agents is the missing control of the agents' (proper) execution. The service center allows tracking of mobile agents in general and thus, addresses the problem of the autonomous agents' execution being transparent to the user (which, in most cases, is regarded as a major benefit of mobile agents). For the particular problem of mobile agents which are unable to continue their execution due to a service or user being unavailable – as with disconnected operations – the kindergarten concept in connection with the agent transfer protocol provides a mechanism to store and revive these agents. What still needs to be tackled is the occurring of failures. In traditional client/server based architectures, an error occurring with a remote procedure call can be taken care of by the exception handling on the client's side. A malfunctioning mobile agent, however, in many cases will not be recognized by its originator at all, especially in the case of disconnection. Therefore, a specific means for treating malfunctioning mobile agents is needed.

Mobile agent migration and execution can fail due to a number of reasons, both on agent level (e.g. erroneous agent code, unforeseen exceptions) and on agent system level (insufficient

resources for correct agent execution, system malfunction). The aim of the maintenance service is to generate reasonable error information in all of these cases and to retrieve the agent state and, if possible, to save the information and results the mobile agent has collected and generated so far. This is important not only because mobile agents can carry sensitive data which they might have collected over a longer period of autonomous actions without user interaction, but also in order to preserve information that helps to determine the cause of the failure which occurred.

A number of components interplay in the maintenance concept. First of all, the agent system containing the malfunctioning mobile agent needs to be able to initiate the maintenance service. This is a critical aspect if the failure occurred on agent system level. If no specific actions were taken prior to a system breakdown, rescuing a mobile agent is not possible, unless it has already been stored persistently. In case of a failure on agent level or a partial failure of the agent system, the procedure described in the following can be initiated. Components involved here are the maintenance service, which is the central component for error handling, the notify service, which has been used as well for the kindergarten service and will help to retrieve the remainder of the failed mobile agent, and the maintenance branch of the directory service.

If a failure on agent or agent system level has occurred, a corresponding exception handling is initiated by the agent system the mobile agent resides at. First of all, an error message is written to the agent directory so that the failure can be noticed and handled, even if the entire agent system crashes prior to further error handling. The directory entry in the maintenance branch containing the agent name and the name of its owner thus resembles a final SOS[1] signal.

In the next step, the agent system will continue collecting the remainders of the mobile agent and sending them to the maintenance service, which can be looked up via the service center. This step is dropped if the maintenance service is executed in the agent system as a default system agent. Although it sounds reasonable to keep the maintenance service as a default system agent in each of the agent systems, providing both alternatives enables more flexibility, in particular if considering light weight agent systems running on PDAs.

On reception of the mobile agent (this is not the case if the maintenance service is a default system service), the maintenance service persistently writes its remainder to a sub-tree of the maintenance branch and adds an entry to the directory which specifies all information required to look up for the agent remainder, namely the agent name, the name of its owner, and the location of the sub-tree from which to retrieve the agent remainder. In addition, the size of the remainder is given. This is valuable in connection with small devices as it allows to decide on the suitability of retrieving an agent's remainder depending on its size. By adding this entry to the directory, the rescue phase is completed.

1. Save Our Soul/Ship

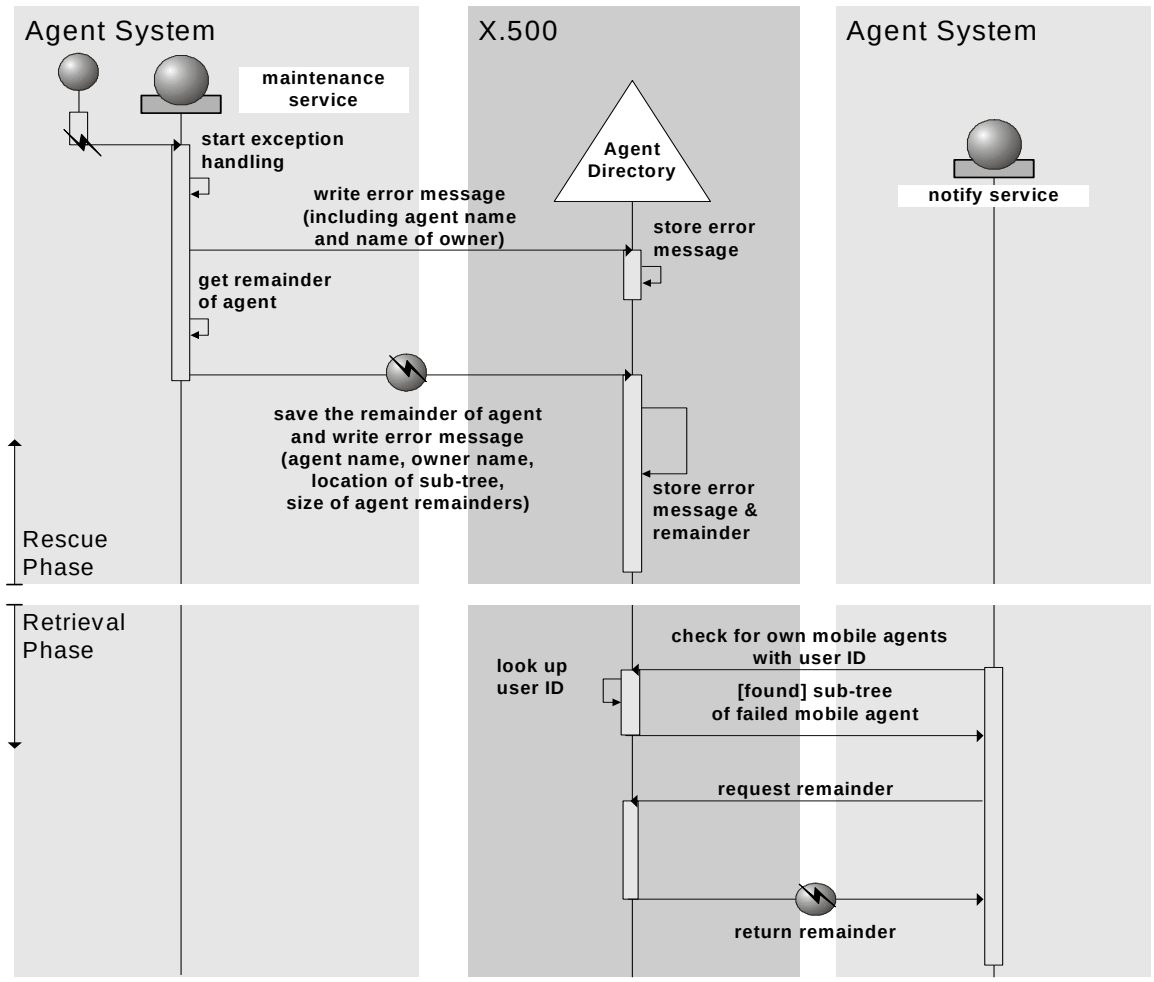

Figure 4-15: Sequence Diagram of the Maintenance Concept

Similar to the kindergarten concept, the reconnecting user retrieves both returning and saved mobile agents by automatically invoking the notify service on turning on the PDA or by manually starting the notify service on any agent based application. The maintenance branch contains all necessary information to either contact the kindergarten service or retrieve the remainder of the failed mobile agent in the sub-tree of the maintenance branch. Thus, the owner of a mobile agent deploys the notify service in order to retrieve the remainder of a rescued agent or collect it from any kindergarten service. The notify service will first check the agent directory for a mobile agent, searching for the user ID or the agent name. If an entry is found in the directory, the location of the corresponding sub-tree is returned in case a failed mobile agent has been saved by the maintenance service. From this sub-tree, the remainder of the agent can be retrieved and analysed for reasons which have caused its failure and for information it contained. The entire process is depicted in figure 4-15.

4.3.3 Remote Invocation of Mobile Agents

It is likely that mobile users activate agents through agent based applications or simply invoke agents through small devices. For this reason, a protocol to remotely invoke mobile agents on the fixed network has been implemented. Depending on the capability of the calling machine, a mobile agent is transmitted with the so-called *launcher* or a mobile agent is executed remotely with the help of the *invoker* service. The launcher has been realized for PDAs that are able to process Java Bytecode and the invoker service is intended for small PDAs allowing only a lightweight protocol.

The Java launcher constitutes a kind of lightweight agent system, which merely consists of the components necessary to send an agent and optionally to receive returning agents. Any application can start a mobile agent by means of the launcher using its application interface. The launcher is initialized with at least one serving agent system in the fixed network and thus, can establish a connection. The mobile agent is delivered to one of the agent systems in the network, which do not need any additional components to handle the agents. The delivered mobile agent can be treated like any other arriving agent, and starts the agent's life cycle to accomplish its tasks.

The support of small PDAs or even simpler mobile phones is realized with the invoker service, which consists of the invoker protocol and the invoker service agent. The idea is to allow mobile users to activate predetermined mobile agents on a pre-set agent system by means of SMS or, more advanced, by means of the *Wireless Application Protocol* (WAP). At the time of establishing the testbed, WAP has been in the specification process, thus could not be considered. By now, it is a de facto standard for providing Internet communications and advanced telephony services on digital mobile phones, pagers, PDAs and other wireless terminals. Within the JAE testbed, the invocation service has been shown based on the *Hypertext Transport Protocol* (HTTP), SMS and the *Simple Mail Transport Protocol* (SMTP).

The task of the invoker service agent located in the fixed network is to accept invocation requests and to execute a mobile agent according to the specified properties. Although the invoker service in the following testbed only accepts the protocols HTTP and SMTP, other application protocols like WAP can easily be adapted. In the JAE testbed, a mobile user may access the agent based services by means of an ordinary web browser or e-mail. All necessary information to initialize a service is obtained by forms in web pages or prepared e-mails. One can imagine that service providers might be interested in offering such kind of web pages or e-mail servers to supply additional services to their clients. In the first approach, the invoker protocol has been kept very simple to be efficient and meet the limitations of the calling device. ASCII-code arranged as key-value pairs describe the agent and its initial parameters, thus the invoker protocol is applicable for SMS.

4.4 JAE Testbed

In this section, the JAE testbed is described following a scenario. The JAE testbed has been set up in cooperation with the department of computer science of the Ludwig-Maximilian University Munich[1] (LMU) and the department of computer science of the RWTH Aachen University. Most measurements have been obtained in the local area network of RWTH. Comparisons have shown that no significant and surprising changes occur when wide area networks are involved.

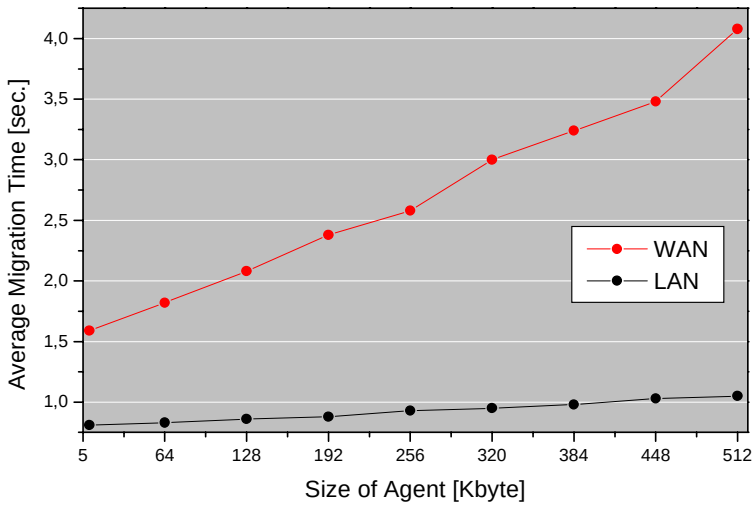

Figure 4-16: Average Migration Time depending on Agent's Size.

The department of computer science at RWTH is interconnected with a 100 Mbps Ethernet, which is connected by a 155 Mbps *Asynchronous Transfer Mode* (ATM) RWTH backbone to the so-called *Breitband-Wissenschaftsnetz* (B-WiN) backbone in Germany. The same holds for the department in Munich. The curves in figure 4-16 reflect the measurements done in the *Wide Area Network* (WAN) and the *Local Area Network* (LAN). Due to the basic properties of TCP, it is apparent that the average migration time of mobile agents rises more steeply than in the local area network depending on the size of the agents. While within the LAN mobile agents with sizes up to 400 KBytes migrate in less than 1 second from point to point, the migration time from Aachen to Munich takes at least 1,6 seconds and could take up to 4,1 seconds.

The testbed has served to show the potential behind the agent based services within a wireless environment. The focus of interest was to demonstrate the support of small mobile devices with different notification services. The combination of the services by means of a mobile agent works fine within a fixed network. The involvement of simple mobile phones has been

1. Thanks to Professor C. Linnhof-Popien for the provision of the machines.

realized with the invoker service. In this testbed, any combination of the following services could have been shown:

- a distributed calendar schedule application
- look up service for train connections
- e-mail filter service
- e-mail notification service
- SMS notification service
- fax notification service

What does it mean? Due to the fact that JAE has been implemented as a service-based agent system, mobile agents can be programmed comfortably to use the services in parallel, one after the other, or mixed. One of the major advantages of agent programming paradigm is the capability of fast prototyping and the combination of existing services. For example, a user can define filters for his e-mails, preferably by means of a browser with a wired connection, and determine at what time a notification has to be delivered via SMS, e-mail, and/or fax. Furthermore, depending on the subject of the e-mail, different tasks can be assigned to the mobile agent.

As described in Chapter 4.3, there are several ways to return the results once the agent performed its task. First of all, the agent can send back the results using services like SMS, fax, or email, but it is also possible that a mobile agent entirely returns to the launcher or the agent system on a laptop. In the latter case, a combination of a notification and the kindergarten service makes sense. If the launcher or the agent system is not reachable, the mobile user can be informed that the agent has fulfilled its task and is willing to return, before the agent is suspended through the kindergarten service.

The JAE testbed includes the GSM/PSTN access network and a switched local area network with access to the Internet. A major focus has been directed at the integration of GSM with the available data transfer capability of 9.6 kbps, and the exploitation of SMS. Figure 4-17 depicts the simplified distributed JAE testbed showing only 4 host machines, each running an agent system and an LDAP directory server acting as the agent directory. The hosts *Q42*, *Worf* and the directory server are SUN Ultra 1 work-stations (167 MHz) equipped with 128 MB RAM. Both, *Frontera* and *Carolus*, are Pentium 200 MHz PCs with 64 MB RAM. The *Carolus* PC is connected to the Public Switched Telephone Network via a 28.8 kbps modem. The deployment of service agents is as follows: the telecommunication service agents are running on the *Carolus* agent system exploiting the modem. For example, a SMS service agent delivers a short message by calling a SMS Centre (SMSC) of the GSM provider and transfers the message via an extended subset of the *Universal Computer Protocol* (UCP) defined for the European Radio Message paging Systems [ERM97]. The email filter service is executed at the *Worf* agent system and frequently checks the mail-box using the POP3 protocol. To allow remote invocation,

the invoker service is located at the *Q42* agent system. Additionally, several mobile agents are stored at this host to be executed by the invoker service such as the email notification service. The *Frontera* agent system offers the look up service for public transport schedules. An additional JAE monitor running at the *Frontera* PC visualizes the movement of mobile agents within this testbed for demonstration purposes.

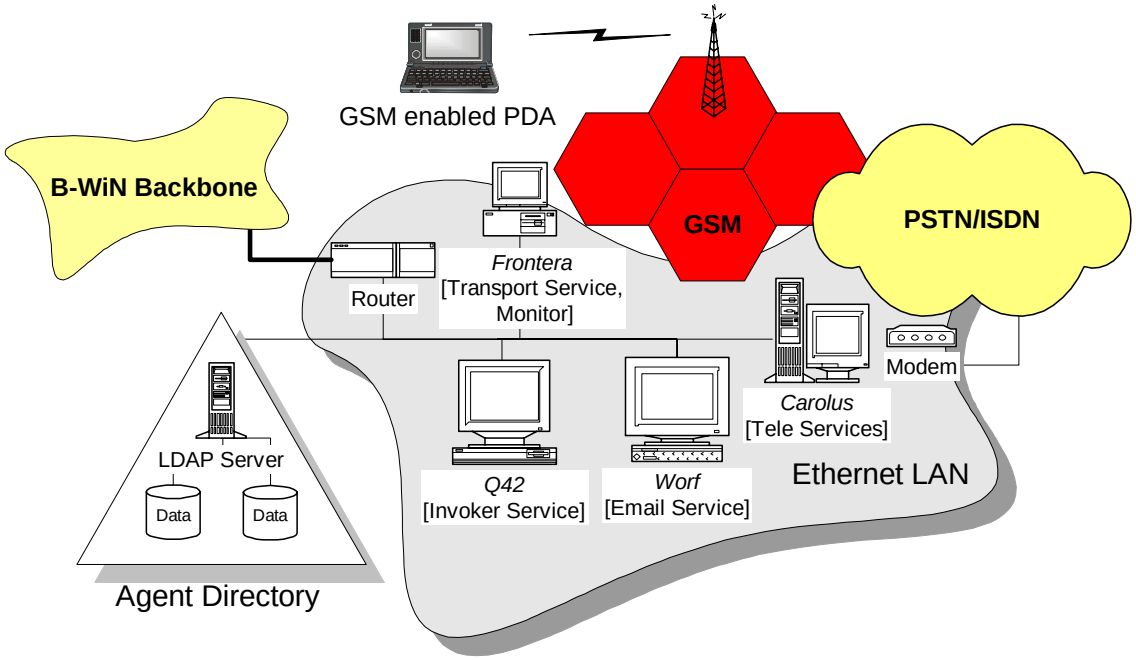

Figure 4-17: JAE testbed at the Department of Computer Science (RWTH Aachen)

Mobile agents can be executed in each of the agent systems manually. In doing so (e.g. at the *Worf* agent system), the mobile agent first requests some parameters from the user that are required to fulfil a task. With this information, the mobile agent subsequently executes its program code and if required, dynamically looks-up services that are stored in the agent directory. To achieve a task, the mobile agent may migrate between all agent systems and return to the starting point with all results.

Imagine a mobile user specifying the work for an e-mail service. This user may enter the parameters in a predefined e-mail form (this form could be delivered to all subscribed clients by the service provider). After filling in this form, the user only needs to return the e-mail to the service provider using any common e-mail client and only a short wireless access to a gateway. The invoker service at the provider filters the parameters automatically and executes a specified mobile agent. The client, equipped with an SMS/GPRS-enabled PDA, is almost always ready to receive the results after the disconnected operation of the mobile agent is fulfilled. The task of the e-mail agent is to monitor incoming e-mails for a specific person and with a defined subject, to e.g. prepare a time schedule of public transportation means, and finally, to send the transportation plan via SMS and notify the user to download the requested

email. If the mobile user is equipped only with an SMS enabled mobile phone, the personalized mobile agent will notify the user using SMS messages that are readable on the display of the device. As an extended service, one can imagine to activate a voice enabled modem to notify the client via voice.

Let us follow the path of a mobile agent after its invocation via the web page or the email form. The JAE monitor displays an invoked mobile agent at *Q42*, where the invoker service agent is located, and receives the demand for activation of the personalised email agent. This agent looks-up the email-filter service and migrates to *Worf*, remaining there until the service agent scanning the user's mail-box has been successful or the waiting time has expired. Upon success, the email agent again migrates to *Frontera* to obtain and calculate an optimized transportation plan and time schedule for the user. Depending on the user's terminal, the email agent simply sends an SMS message or prepares a MIME coded message, which in turn will activate the PDA being in suspended mode. As soon as an SMS daemon process detects a message, the program checks which applications to start. Each user has to pre-define the individual security restrictions and what kind of automatic processes are to be executed without user intervention. In the prototype example, the PDA has a configuration that allows an email application to establish a connection to the email server and download the requested and specified email, and to notify the user accordingly [PaRe98]. This simple example shows how the combination of mobile agents and service agents may support mobile users. One can imagine that in the 3G mobile system, GPRS and Java Virtual Engine enabled devices accept mobile agents, so that SMS notification is no longer necessary.

Size of mobile agents	5 Kbytes
Size of e-mail form	~ 2 Kbytes
look up time for agent services	160 ~ 1060 ms
Serialisation of mobile agents	200 ms
SMS notification after delivery	> 20 sec.

Table 4-18: Transaction Times and Data Sizes

Table 4-18 gives a feeling about the transaction times and the minimum size of mobile agents for the presented agent environment. The measurement values show that the migration flow of mobile agents is fluent. It has turned out that time delay during information retrieval - due to network load - or the load of the SMSC severely affects the overall execution time of the mobile agent rather than the Java Virtual Machine. The total time, from detecting the awaited email by the filter agent service to reception of the SMS message by the mobile phone, is less than 1 minute, depending of course on the load of the SMSC and the network. In general, however, the execution time strongly depends on the size and the task of a mobile agent and is not really relevant for disconnected operations in the sense as presented in this scenario.

4.5 Reflection of Agent Technology in TINA

Results achieved so far have emphasized the importance of mobile agent technology for mobile computing. In the following, a brief outlook of agent technology with focus on the *Telecommunication Information Networking Architecture* (TINA, [TINA97]) is discussed. So far, the TINA Service Architecture does not provide satisfactory solutions for supporting nomadic communication. For example, TINA has been designed without considering the importance of future mobile cellular networks like UMTS. This does not only concern aspects of location management and terminal mobility, but also user profile presence in foreign domains.

The provision of user mobility will offer customers personalized profiles, lifelong personal IDs, and consistent services access, independent of the devices in use and the network or the service provider they are attached to. Already today, a multi-provider environment exists and a homogeneous network cannot be expected. Quite the reverse, forthcoming a convergence of the world-wide heterogeneous networks towards UMTS is assumed. In UMTS, the consistent service access irrespective of the terminal type and the access network is planned with the *Virtual Home Environment* (VHE, [VHE00]). The VHE is a capability for providing operator-specific to end users with a consistent look and feel which is independent of location and serving network. It is to facilitate service adaptation to different network environments supporting directly connected, cordless and cellular access. But service logic and bearer control of conventional fixed and mobile networks, which are nowadays based on intelligent networks, will not be able to meet the requirements as they are imposed by the VHE. For this reason, the world's leading providers and manufacturers have joined forces in the *TINA-Consortium* (TINA-C), aiming at the definition of an extensive framework that copes with the strong requirements. The resulting TINA framework takes advantage of object-orientation and distributed systems and is thus seen as an alternative approach to overcome heterogeneity, but also to establish new paths towards fault tolerance, scalability, interworking, and rapid service creation and deployment. However, initiatives of TINA were launched at a time when the importance of nomadic communication in general and the popularity of mobile cellular networks in particular were not foreseen. As a consequence, TINA has ignored aspects resulting from terminal mobility and has only a limited view on the requirements of personal mobility. To cope with these problems, TINA has launched several auxiliary projects which deal, for instance, with VHE [AbPe99] or next generation mobility. Additionally, the ACTS project DOLMEN [WLR98] and the EURESCOM project P608 [Cla98] have been involved in these issues. Independent from the TINA specification a mobile agent middleware handling the personal and terminal mobility is discussed in [BCS01].

In the following, a proposal for the mapping of JAE to TINA is outlined. It is shown how an agent system can be used for supporting and even realizing TINA-based services and how to integrate mobile and system agents into the TINA service architecture. According to the TINA framework, future telecommunication networks can be seen as huge distributed systems with

millions of interacting objects for the purpose of service provision as well as for control and management issues. As such, TINA-compliant networks must be based on a middleware, the so-called *Distributed Processing Environment* (DPE), upon which the various objects are executed. The DPE cares for the provision of transparency mechanisms, like location or distribution transparency, or the overcoming of heterogeneity. Additionally, it is supported by a number of infrastructure services, e.g. a factory for controlling the life cycle of objects or a trading service for locating objects on remote hosts. The DPE itself is based on a kernel transport network for the exchange of invocations performed on the operational interfaces of objects.

The *Common Object Request Broker Architecture* (CORBA) of the OMG is often seen as the most promising candidate for the realization of both the DPE and its infrastructure services. The predominance of CORBA in the TINA context is based on the assumption that future networks will be composed of heterogeneous hosts with different operating systems and applications which have been realized with different languages. However, the advancement of Java in the telecommunication domain as well as the availability of Java for a diversity of platforms also permits another conclusion: the deployment of mobile agent technology, as it is enabled e.g. by the JAE system, is a favourable option or extension. To prove the suitability of JAE for TINA, Table 4-19 provides a mapping between the components of both frameworks.

	JAE	**TINA**
Infrastructure Components	Agent System	Distributed Processing Environment
	ATP/ACP	Kernel Transport Network
	System Agent	Factory
	Agent Directory	Trader/Broker
	Kindergarten Service	
	Maintenance Service	
Service Components	Mobile or System Agent	User Application / Agent
	Mobile Agent	Provider Agent
	System Agent	Service Session Manager
	Mobile Agent	User Service Session Manager
	System Agent	Subscription Manager
	Mobile Agent	Subscriber Agent

Table 4-19: Mapping JAE Components to TINA

The table is subdivided into a mapping of infrastructure and service components. The DPE is represented by the JAE agent system. The agent manager is the most important part in this system. With the mobile and system agent handling, it contains an element for realizing the TINA factory. The monitoring component guarantees that objects, especially if they have their origin in foreign domains, behave in the intended way and do not violate the provider's policies. The TINA framework focuses on a separation of addressing matters by introducing a dedicated stakeholder role, the so-called broker. The agent directory is an ideal platform for realizing this

new stakeholder; due to the choice of X.500 as underlying database platform, huge broker federations and global location transparency between any number of agent platforms becomes possible.

Service components are either represented by system or by mobile agents. TINA follows the deputy principle, i.e. interacting domains maintain agent objects in the foreign domains. These agent objects may then interact with local resources on behalf of the customer, for instance, to negotiate service-usage conditions, to control the keeping of contracts or to set-up a trusted relationship between domains for further interactions. Examples for these objects are the *User Agents* (UA) in provider domains or the *Provider Agents* (PA) located in user domains, i.e. in the (mobile) terminals as depicted in the following figure 4-20.

The UA contains the user's profile and may allocate resources according to his profile. Another user-related agent is the *User Service Session Manager* (USM), which represents and holds the context of a party and supports the suspension and resumption of the party's participation in the service session. In contrast to these objects, the PA acts on behalf of the provider in the end-user device and is used to support the user accessing his UA and setting-up a trusted relationship in this way. Typically, these objects are optimal candidates to be realized as mobile agents, which is due to the fact that their residence may change frequently. Thus, it is not only desirable to support the system with suitable migration mechanisms, but also to expose these foreign objects to adequate security and resource control mechanisms as they are provided by JAE. On the other hand, local objects focusing on service provision, like the central *Service Session Manager* (SSM), need access to local resources and are only managed and configured by provider authorities. Therefore, system agents can be used for realizing this kind of components. Often, the level of changes permitted to a user may be limited to the "look and feel" of the working environment, the personal address book, or the configuration of some value added services granted to him. On the end-user level, the VHE is mainly composed of three types of mobile agents:

- one UA, which contains and carries the user profile,
- one or several USMs, each holding the current state of a session the user is participating in, and
- one or several user applications, each being the front-end of a particular service the user has access to.

It is this end-user level that may profit from mobile agent technology most of all. To draw a conclusion, mobile computing profits from mobile agent technology in various ways. It enables users to request and establish their individual front ends on any terminal and from any provider. The agents automatically care for adapting the graphical user interface of front ends to the capabilities of the used terminal. Furthermore, user data, e.g. user profiles, can be managed in a flexible and adaptive way. Customers can run their own deputies in the provider domain that allocate resources or execute supplementary services according to the user profile.

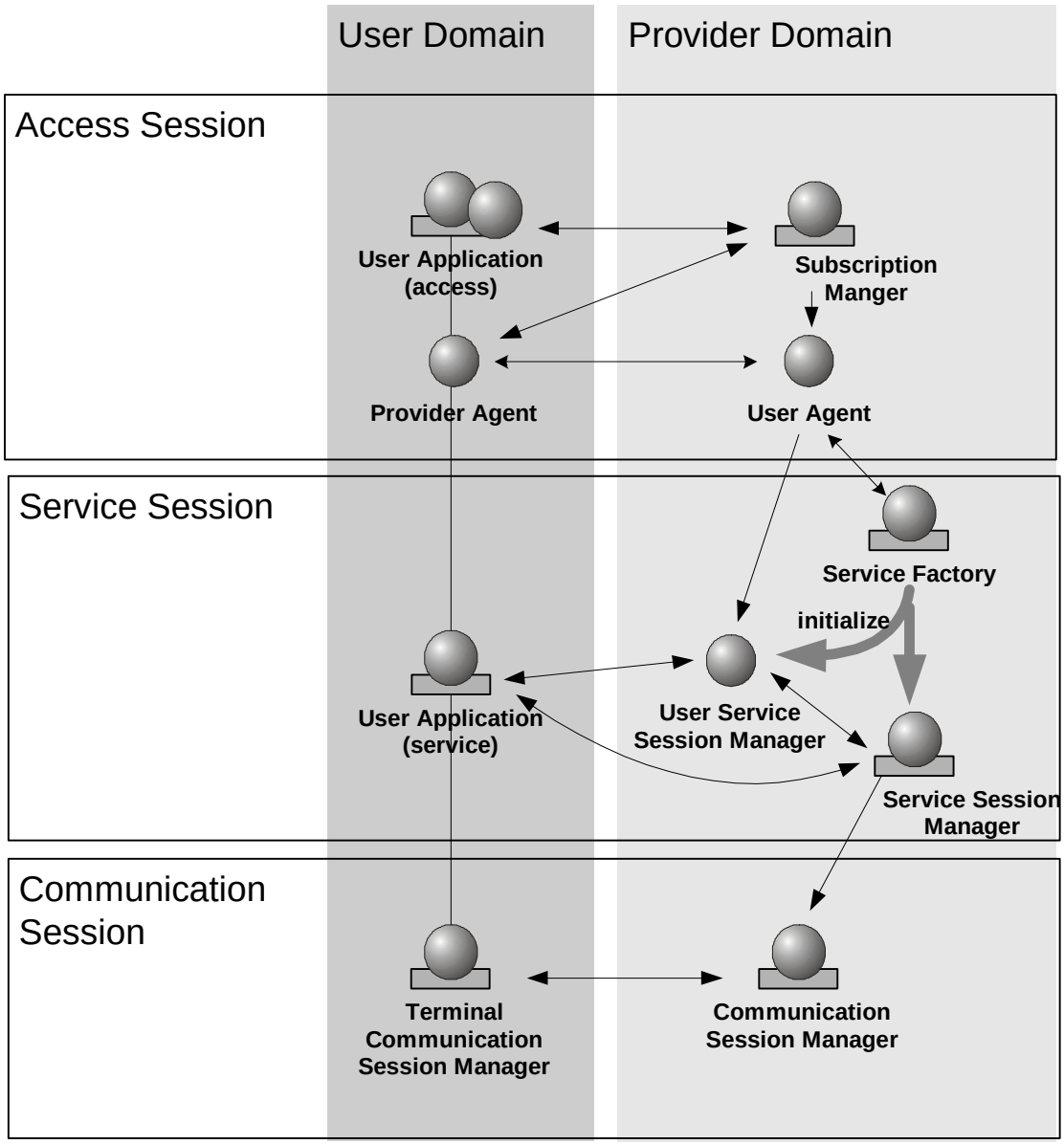

Figure 4-20: TINA Objects visualized as Mobile and Service Agents

Besides this, session mobility is supported in that the participation of a user can be kept, even if this user is changing the terminal, the provider, or even both of them. From the provider's point of view, agent technology helps to arrange user-related resources near the end-user and to avoid long signalling paths through the network. The publications in [LNPS97], [KüPa98] and [KüPa99] provide some analysis in this field and show pros and cons of the various strategies with system and mobile agents. Within the foreseeable future, the research in agent technology will have an effect on new services and on the way they will be offered. This technology complies with the vision of entirely packet-based telecommunication networks and has the potential to be incorporated into the future "intelligent" networks.

4.6 Conclusion

Summarizing, the service center, the kindergarten service, and the maintenance service enable a tracking and handling of mobile agents both during their scheduled execution, and in the presence of unforeseen errors and failures. In this chapter, new concepts have been presented which support mobile computing with an agent system, addressing the problems resulting from disconnected operations. Additionally, the invoker service allows the involvement of very small mobile devices with only limited capacities. Finally, the launcher offers an API to embed agent support in any Java application so that these programs can be simply extended and gain from the mobile agent technology. All these concepts provide crucial support for an agent-based approach to mobile computing, which has been demonstrated with prototype implementations and the demonstration within the JAE testbed. It has shown that mobile agents are predestined for personal disconnected operations, especially in wireless access networks. Therefore, this chapter ends with a reflection of agent technology in TINA and shows a possible solutions based on JAE. Prototype implementations and concepts worked out with JAE have also successfully been adapted to the European sponsored ACTS project *Agent-based Mobile Access To Multimedia Information Services* (AMASE [AMA99]). A banking application based on the agent system demonstrates the agent system's suitability for mobile computing and shows the opportunity for new value added services [CKO+99]. Details about the project and the AMASE agent system have been published elsewhere [LiPa99, PLKS99].

Major experiences gained during the implementation phase include the easy realization of agent-based distributed applications and the flexibility that the mobile agent concept offers to application programmers. Once the set of service agents has been defined and established, the combination of these services to fulfil tasks by means of mobile agents is simple. The service-based agent programming concept, introducing the different agent classes with system and mobile agents, shows the advantage of JAE's architecture. Further work remains concerning examination of the feasibility of interaction with mobile users over GPRS and the adaptation of WAP to the invoker service. Obviously, many unforeseen and new value added services can result from the merging of the agent technology, the Internet, and the telecommunication services, which remains as a challenging work to investigate.

Chapter 5

Agent-based Application Modelling

In this next to last chapter, the pros and cons of an agent-based application in the context of network management is shown. Mobile agents have been identified to suit just well the needs of network [PWW00] and telecommunication management [SSS+99]. In this frame of reference another field of research is the management of the Common Object Request Broker Architecture (CORBA) with agents [LiPa98].

In order to show the efficiency of the agent technology, the focus here is on analytical models of different agent-based network management concepts. Again, turned off and not reachable devices in wireless access networks (e.g. wireless LANs) are particularly considered. Comparisons are drawn with the agent-based management approach and the *Simple Network Management Protocol* (SNMP), which is the prevailing conventional management concept and follows the client/server paradigm. It can be shown how quantitative improvements of crucial performance parameters can be achieved including reduced network traffic and response times. It turned out that a mixed mobile agent and SNMP strategy is an efficient approach for network management, especially if frequently switched off devices (PDA, Smart Phone, notebook, etc.) are involved.

Following, the network management approaches are presented. New strategies based on agents are discussed and their suitability for meeting current and future network management requirements is assessed. Mobile agents seem to be a solution for nowadays typical heterogeneous network organization with its large and growing architecture including a variety of LANs, WLANs, and WANs, supported by bridges, routers, and distributed computing services and devices. The main part of this chapter are the analytical models of the different management strategies regarding both packet loss probabilities of the network and failure probability of the network devices. It becomes more and more important to consider these failure aspects due to the fact that wireless links and dynamically changing network topology belong to today's networks.

5.1 Network Management

Network management aims at planning, surveying, and coordinating resources of computer and communication networks to ensure availability and reliability of the systems and services involved. The *International Organization for Standardization* (ISO) Management Framework Standard [X.700] says: Network management provides "*the facilities to control, co-ordinate, and monitor the resources which allow communications to take place in the Open Systems Interconnection environment.*" The structure of both networks and systems, however, is changing dramatically, and the overall complexity is rapidly increasing. The main reasons for consideration of mobile agents in this area include the proliferation of wireless links, dynamically changing network topology, and the heterogeneity caused by often incompatible multi-vendor environments. Along with the complexity of the underlying networks, the requirements on management solutions are increasing. The functional tasks of configuration, fault, performance, security, and accounting management specified by the *Open Systems Interconnection* (OSI) have to be further enhanced to address the multitude of management tasks. Currently, two main protocols for system and network management coexist, SNMP and the *Common Management Information Protocol* (CMIP). Both are applied with a central management station in a client/server environment.

The central management station invokes operations on so-called managed objects that represent each resource to be managed. In both environments, Internet and OSI, the database containing these objects is referred to as a *Management Information Base* (MIB). The MIB is a structured collection of objects, which reflect the status of the managed resources at a node. Both protocols provide, in the term of OSI, *services primitives* for exchange of management information between the two entities (manager and managed object). The network manager application uses request and operation command primitives, whereas the process that manages the objects uses primitives for responding and reporting events. However, there are fundamental differences between the two management protocols. Based on the OSI Reference Model, one of the *Application Service Elements* (ASE) of the application layer defines the *Common Management Information Service Element* (CMISE), that covers the network management functionality and is specified in two parts:

1. The *Common Management Information Service* (CMIS) defines the interface with a user, specifying the services provided.

2. The *Common Management Information Protocol* (CMIP) specifying the *Protocol Data Unit* (PDU) and the associated procedures.

CMIP offers a rich set of protocol operations on both manager and managed object side. In addition to service primitives for creating, accessing, and deleting managed objects, CMISE also defines services such as filtering and scooping. In accordance, the MIB is more complex and follows the object-oriented principles offering advantages of encapsulation and inheritance

of the managed objects, rather than consisting of simple variables as in SNMP. The application accessing the MIB (agent[1] process), by means of CMIP, may do a far more extensive preprocessing of events and information filtering than the Internet counterpart could do. However, the complexity of the OSI approach has been one of the main reasons why only few CMIP implementations have become available until today, and the management protocol has found very little acceptance [Ram98]. In the following, the focus is on SNMP that is more relevant for the comparison with the TCP/IP based agent management strategies.

The SNMP specifications were first issued in 1988 by the Network Working Group of the *Internet Engineering Task Force* (IETF) and soon became the most widely used management protocol [RFC1065]. SNMP is intended to operate over the *User Datagram Protocol* (UDP, [RFC768]) and is used by the *Network Management Station* (NMS) to achieve network management. The network management architecture includes the SNMP, the NMS, the *Network Management Agent* (NMA), and the MIB. Consequently, each agent of a managed device must implement SNMP, UDP, and IP, furthermore, it controls the MIBs. The intelligence of the management is centred in the management application of the NMS, which uses following key capabilities of SNMP to control and exchange information:

- *Get*: enables the NMS to retrieve values of managed objects
- *Set*: enables the NMS to set values of managed objects
- *Trap*: enables the NMA to notify the NMS of significant events

Typically, the NMS retrieves all relevant information from the managed objects polling the NMAs. Although, this can become impractical or even infeasible if all the NMAs are to be managed containing a large number of managed objects with a large number of values. The trap-directed polling may partly free the NMS from continuously polling and thus, reduce network load, if some baseline performance statistics or average numbers have been collected. Once some limits are known, the NMS can refrain from polling and instead, each NMA is responsible for notifying the NMS of unusual or significant events. The overall principle of the SNMP management architecture is shown in figure 5-1.

There are no specific guidelines in the standards as to the number of NMSs or the ratio of NMSs to NMAs, but in general, SNMP is used in a traditional centralized network management scheme. The practice has shown that for the reason of redundancy, a second NMS is used in a backup role, but not for distributing the burden of the central NMS. Thus, the first version of SNMP does not meet the requirements of a large distributed system, despite being simple to implement and use. Due to the lack of NMS-to-NMS communication, the inability of transferring a bulk of data, and the lack of security new versions have been issued addressing these deficiencies, namely SNMPv2 in 1993 and the current version SNMPv3 in 1998. The main improvement in SNMPv2 is the support for a decentralized hierarchical network management

1. client process in the context of network management

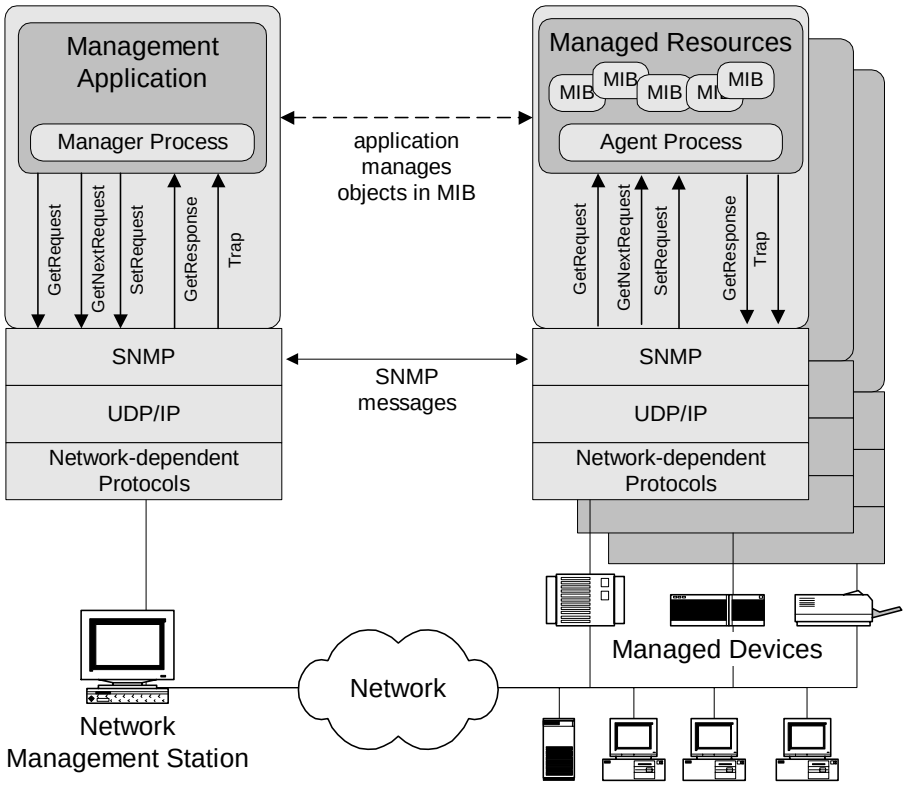

Figure 5-1: Network Management Architecture based on SNMP

architecture. The introduction of multiple top-level NMS and the launch of intermediate managers allow a better load balancing of the processing burden and reduces the total network traffic. The reason is that each of the top-level NMS might directly manage a portion of the total pool of agents, but might also delegate responsibility to intermediate managers that, on the other hand, monitor and control the agents under their responsibility. The most substantial improvement of SNMPv3 are the security features. The modules added to this version take care of message creation and parsing, access control, handling of data, and finally the authentication and encryption. A detailed discussion of the evolution of SNMP and the protocols can be found in [Sta98].

5.1.1 Management by Delegation

Centralized network management is obviously the simplest approach that has gained wide acceptance. It has been apparent, though, that further improvement is indispensable. The deployment of the traditional approaches has shown that network management with a central manager results in a low degree of flexibility and scalability of the managed network. The problem becomes more conspicuous with intermittent network connectivity that is the case including wireless access networks. The first version of SNMP caused a lot of data traffic for transferring management information, due to the limited possibilities of the physically distrib-

uted NMAs and the complete management logic residing in the central NMS. Admittedly, the following version of SNMP has well considered the above mentioned drawback, but the environment is still built around a central manager. This architecture incorporate several factors which make such an approach not forward-looking, as systems grow increasingly complex and change topology dynamically. It is not certain that future networks will have a hierarchical topology, thus the static distribution of the intermediate managers will be sufficient.

It has emerged that the amount of monitored data to be processed in real time is increasing drastically. On the other hand, the data overhead by means of network management has to be avoided and must not become a critical factor. Separate attention is needed for mobile devices deployed in a wireless local area network, because they may cause supplementary management data traffic due to repeated disconnections. So far, the network management has not taken into account the wireless access network with its special problematic nature for management application. In such an environment, extra notifications will increase the management traffic deteriorating the overall throughput of the network and possibly generate congestions. Even assuming that the bandwidth of the network outgrows the traffic caused by the management application, a huge amount of data must be processed. Although, intermediate managers cushion the load of the central NMS, the number of manageable objects by one manager is confined; and the insertion of a new level of intermediate managers is very lavish. Certainly, the NMAs free the managers from a bigger flood of information and reduce the network load by pre-processing information and only throwing notifications if needed. However, the functional means of an NMA is limited, and trying to consider all contingencies in advance is not possible with regard of the required code and the growing complexity of the managed network. A huge system may contain countless parameters to be observed and with its continuously changing demands, different functionality is needed from the NMA. For example, mobile computing is becoming an important part of today's networks. In client/server-based approaches, such as SNMP and CMIP, this problem of intermittent connectivity is not addressed. The overall conclusion, therefore, is that a more flexible approach is needed. In the following, the advances of the mobile agent technology is discussed.

The role of network management agents has to be enlarged. Beside the tasks of data collection, pre-processing, and reporting, the quality of mobile agent technology is to exploit to comply with the demand for a flexible and distributed management. With an agent-based manager application, a dynamic delegation of tasks can be realized. Depending on the requirements of the work, newly created mobile agents can be sent out that follow their work schedule autonomously. In this way, the agents take management load of the manager and continue in the disconnected mode without a direct link to the manager. The problems raised by intermittent network connectivity are handled by the autonomy of agents and do not provoke a failure treatment as with traditional client/server systems.

An interesting network management approach deploying mobile code has been published in [Lip00]. In contrast to the remote programming paradigm, the so-called *remote evaluation*

only allows to execute recently sent object codes on different physical machines. However, such a distributed agent network, which allows to execute code sent by a network manager, provides a simple facility for code updates. Advantages arising from remote evaluation over the traditional client/server approach are always given when a bigger amount of values has to be collected and frequently changing calculations have to be executed. In the client/server technology, functions for the computation are either implemented in the server or at the client side. Having the whole function at the server side intends that data always has to be transferred to the server, which burdens the network traffic. On the other hand, the implementation of functions at the client side distend the code, because all functions have to be implemented beforehand, and a lot of management overhead is caused; if only one new computation function has to be added. The network administrator would have the awkward job to reinstall all new clients. Remote evaluation, however, allows to dynamically update new functions to the stationary agents/agent system by just transferring the code. Depending on the amount of data to be collected and processed, a certain kind of load balancing and distribution of calculation is reached. Moreover, a reduction of the network traffic is gained by delegating the tasks and merely transmitting the results of each agent.

The model with the concept of *elastic servers* is going a step further. The idea is to have a language-independent management application based on remote evaluation. Thus, the core feature of the elastic server is its ability to support translation and dynamic linking of delegated agents. In this model, each agent is executed by an elastic server at the destination host. Both agents and elastic servers are realized as multi-threaded processes to permit dynamic modifications on execution. A proprietary delegation protocol ensures a proper transfer of the agent's code to the elastic servers and its execution. A more detailed description of management by delegation can be found in [GoYe98]. So far, it has become evident that management by delegation is a valuable alternative to the traditional centralized management. It offers a high degree of flexibility and scalability, addressing the problems of optimal bandwidth usage, dynamic code adaptation for process optimization, and continuous handling of management functionality even in wireless access networks with more frequent link interruptions. However, one important issue not considered in this approach is the autonomy of agents and their ability of migration, which is the subject of the next section.

5.1.2 Network Management based on Mobile Agents

Applications like the network management can gain advantages by means of the mobile agent technology. The typical characteristics of agent technology such as the ability to communicate with peers or even with users, their feasibility to migrate, and the capability to pursue tasks on behalf of other applications can be favourably employed. In an agent-based management application, the network manager is in charge of scheduling the mobile agents; i.e. on the basis of some control information, the manager can delegate mobile agents with management operations, though is not limited to only one task and location. The manager injects the mobile agent

into the network equipped with specific management tasks, which the agents are going to fulfil autonomously. Sent out agents can be controlled with the agent communication facility, and the replacement of them is possible any time, granting the demanded flexibility in network management and the easy update functionality. Because mobile agents themselves are independent entities running on any agent system, they suit very well into the fields of intermittent network connectivity (see chapter 4). While trying to accomplish their goals, they operate disconnected from the manager and only establish a connection when an exceptional situation arises. Sensibly, the exchange messages and protocols are optimized to relieve the network traffic. Depending on the network management approach, the protocols have different effects on network load, which will be shown in chapter 5.2 by means of selected management applications.

The new feature of autonomous execution of management operations and the dynamic adaptation and reaction depending on the network environment brings out new archetypes for network management. In this context, management agents are initially equipped with a set of functions and are deployed into the network proceeding with the management operations. At the same time, the execution path of these mobile agents is not foreseeable, but rather depends on the current state of the network and its devices. With mobile agents, the semantic routing is realized because dynamic adaptation and varying of the task schedule are assumed in case changing conditions such as interrupted communication or switched off devices are encountered. The management agents are not only restricted to gather and evaluate network status, but have the possibility to further acquire and deliver information about management activities. Information exchange with peer agents, the manager agents, and even the user concerning the managed device may be valuable for the agent's confirmation in strategic and critical decisions. Certainly, the interaction with the environment and other agents, by means of the agent communication facilities, is an important feature for autonomous work.

The aspect of autonomous work always lets assume that some intelligence in the agents has to exist. The research in *Artificial Intelligence* (AI) is well discovered in comparison to the mobile agent technology - and today, the merging of both technology is an ongoing research topic. Already the decision of whether a mobile agent should migrate to a new location or not may contain some basic calculations and intelligent assistance in reaching a decision. One has to consider that the migration process creates a considerable overhead with all the processes a mobile agent has to stride through, namely packing the state and data, compression, authentication, serialization and de-serialization of the agent, and many more steps. Depending on the amount of data to be processed, the migration can be more costly than calling a remote procedure. In principle, a mobile agent should be kept small and restricted only to a few KBytes to minimize overhead deploying them. Especially in network management, small agents are prerequisites, despite the fact that the management intelligence is required for the tasks the agent has to fulfil. In [PMM89] an approach is presented that is constituted on an expert system for network management. In this system, agents are able to assert rules to the management knowl-

edge base and to infer conclusions from the given rules. The main drawback of this approach, however, is that in general expert systems are centralized. Even with a decentralized system the applicability and costs of this approach with mobile agents must be weighed. Unquestionably, mobile agents asserting rules and obtaining inferences from expert systems may rely on a good solution, and also free the manager from explicit interactions, but on the other side, accessing the expert system creates additional communication traffic.

Another interesting approach transferring management intelligence from the manager to mobile agents is yielded by a taxonomy of the agents into several classes. All these distinguished mobile agents organize together the so-called *swarm intelligence*. Basis of this model are agents that migrate to the network nodes and survey relevant parameters of the managed objects. Another class of agents continuously circulates the network and analyses and evaluates the data given by the above mentioned basic agents. Assessed values packed in a message are left at a node to serve the additional class of purposeful working agents. On the basis of the messages, the delegated agents can locate failures in the network or devices and achieve their concrete task. Detailed descriptions of the agent taxonomy and the swarm intelligence are given in the references [WPD02] and [MLZ03], but as with the use of expert systems, many questions still need to be addressed.

Most of these new concepts based on mobile agents are in the prototype stage. Although, they promise to enhance network management with new values, a verification of performance improvements has not been considered in detail. In the remainder of this chapter, a comparison of the traditional, centralized management paradigm and the new approach based on mobile agents is given. The comparative evaluation of SNMP management and the mobile agent-based management approach have been carried out on a packet-based level by means of analytical modelling.

5.2 Analytical Modelling of Management Applications

It is undisputed that agents have advantages, thanks to their ability to work autonomously and dynamically, such as flexibility and concepts for fast programming. However, these qualitative characteristics in general cannot be expressed through simple figures. Thus, concrete and determinable values have been searched for to give a clear argument for benefits of mobile agent deployment. With the help of the management application, a precise state of the network traffic in bytes and bits is given. In this section, an in-depth mathematical model analysing the management traffic on the network is discussed, taking into consideration both the loss probability of packets in the network, and device failures. In order to support network administration, the analytical model will help to decide if the agent-based paradigm should be selected instead of the traditional client/server architecture. The model is based on the assumption that the number of devices and the average failure of both the network and the network devices

(routers, work-stations, notebooks, or PCs etc.) is available or can be estimated. The calculations will compare three approaches, a purely mobile agent based network management strategy, a combined approach with mobile agent using SNMP in the local area network, and the traditional UDP-based SNMP.

So far, packet loss probabilities of networks has been not considered in detail. For example, the Carleton University has intensively examined mobile code for network management using their Mobile Code Toolkit to automate the management activities of communication networks. The overriding goal was a self-configuring network that automatically recognizes and includes new resources [ElBi99]. Unfortunately, analysis was never subject to many publications. Likewise, the very detailed thesis of G.S. Goldszmidt [Gol96] comprises different performance analyses supporting the advantages of mobile agent based network management, but does not consider network failures. Another example is the JAMES Platform, a generic management model that focuses on support for remote installation and maintenance of the infrastructure, well defined boundaries between applications and the supporting infrastructure, and stronger integration with already installed management systems. Again, the prototype implementation shows the advantages of the agent technology with its cost-effective infrastructure management, clear separation between infrastructure, mobile agents, applications and users, remote management services, and the integration with legacy network, but network failure has not been considered [SSF00]. This, however, is required due to the potential packet loss of UDP, as well as the re-transmission algorithm used by TCP to recover from erroneous transmission. Furthermore, the increasing proliferation of wireless LANs and dynamically changing network topology demands re-calculations of management traffic, taking into account the probabilities mentioned above. Moreover, it is necessary to consider the different possible states of managed devices or objects. Given current practice, personal computers are often switched off at night or during business trips and vacations. In addition, notebooks and mobile devices in wireless access networks are frequently disconnected or change their locations. This aspect must not be neglected, as the network management station is not able to distinguish between a device being defective or switched off.

In the following, network management based on SNMP and mobile agent is discussed. Figure 5-2 depicts the three analysed network management strategies in terms of component participation, distribution, and flow of information between them. In the agent based network management approach, mobile agents regularly pass all network devices to collect and evaluate information. The third approach is a mixed strategy; a mobile agent migrates to a dedicated host of the local area network to poll (with SNMP) all devices. The figure helps to understand the analytical model that is restricted to the analysis of network data imposed by the respective management application. In the model, only network and device failures are considered to determine the network traffic overhead caused by the management. The delays because of requests or replies, application size, or robustness are not regarded.

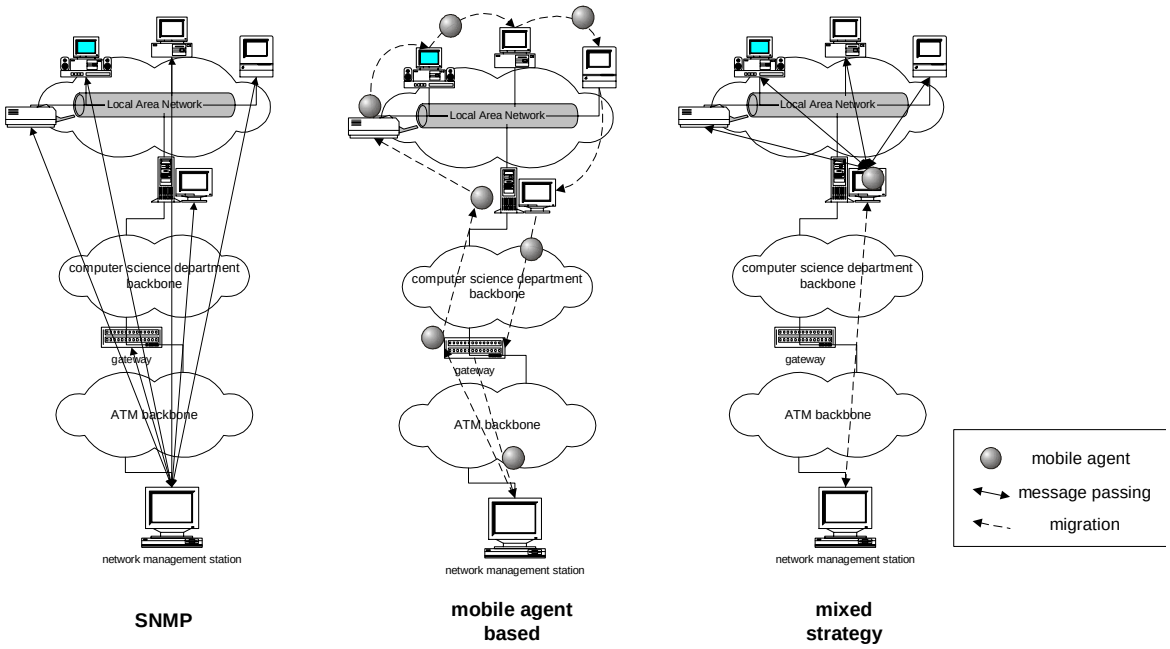

Figure 5-2: Network Management Strategies

5.2.1 Network Management based on SNMP

In the analytical study of Baldi and Picco [BaPi98], the data traffic caused by the SNMP-based management is summed up as:

$$V_{SNMP\_BaPi} := \sum_{d=1}^{D}\sum_{i=1}^{I}(\phi_{SNMP}(P_i)\cdot P_i + \tilde{\phi}_{SNMP}(R_{id})\cdot R_{id}) \qquad (5.1)$$

The calculation is easy to understand. D is the number of manageable devices and I gives the number of inquiries. P_i represents the size of the packet in the i-th inquiry and R_{id} the packet size of the i-th response of the d-th manageable device, both multiplied with the function ϕ, which results in a number bigger than or equal to 1, adding the overhead of the underlying protocol (assuming different protocol overhead of message request and reply). The sum of the request and response packets forms one communication process. The total data traffic is given as the sum of all inquiries over all manageable devices. This formula allows a precise analysis, but requires detailed information on each inquiry and response packet, i.e. a vector of packet sizes for each inquiry and a matrix of response packet sizes depending on the responding network device. It is difficult to obtain these values from a network with e.g. 350 IP-sub-networks and several hundred devices each (as is the case with the network of RWTH Aachen University, which is used as a reference) - even without considering the various potential request messages. The multitude of possible parameters had to be reduced for our purpose, on the one hand, to simplify both model and simulation, and on the other hand to add device and network

failure probabilities. There are two possible solutions to obtain the average network traffic. First, the packet size of all NMS requests are considered and multiplied by their distribution. Then, the average packet size of the inquiry is inserted into the formula. Thus, the following average amount of traffic at the NMS is calculated:

$$\overline{V}_{SNMP} := D \cdot I \cdot (\phi_{SNMP}(E[P]) \cdot E[P] + \tilde{\phi}_{SNMP}(E[R]) \cdot E[R]) \quad (5.2)$$

given P and R as random variables for request and response, respectively, with their distribution.

A second option – which is the one chosen here – is to count the average number of status requests to a manageable device from the NMS, taking into account the network and device failures. Depending on network or device failures the NMS repeats the inquiry to the manageable device several times after a given time out. The average number of status requests is multiplied with the packet size of the status request to get the average amount of traffic from the NMS to the manageable device. In the second step the average number of responses and the average amount of network traffic from the manageable device to the NMS is calculated, considering the failures. Before calculating the average number of inquiries and responses, the packet size of a status request and response is observed.

The MIB size of a common status request including a SNMP password is 28 bytes, the response size to such a request is 33 bytes. The protocol overhead of IP, UDP, and SNMP is 60 bytes altogether for both response and request because fragmentation is not necessary for this packet size. Figure 5-3 illustrates the packet size of 88 bytes for a status request ($PS_{request}$) and 93 bytes for a response ($PS_{response}$).

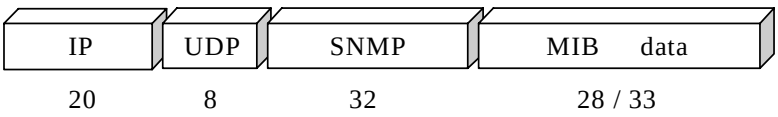

Figure 5-3: SNMP Network Interface Request

The total traffic V'_{SNMP} caused by the status requests of the NMS to D manageable devices without failure probabilities can be simplified to:

$$V'_{SNMP} := V'_{request} + V'_{response} = D \cdot PS_{request} + D \cdot PS_{response} = D \cdot 88 + D \cdot 93 = D \cdot 181 \quad (5.3)$$

In the following, the integration of network and device failure probabilities is discussed. The goal is to calculate the expected total number of status requests delivered by the NMS due to failures. With this expected total number of NMS-requests the depending requests and

responses in the sub-networks can be calculated and finally the expected total traffic in the network.

The probabilities of network and device failures are defined and are distinguished between different sub-networks as follows:

$$\begin{aligned} &SN := \text{number of sub-networks} \\ &f_{net}(i) := P(\text{failure of sub-network } i) \quad , i = 1,..,SN \\ &f_{device} := P(\text{device failure}) \end{aligned} \tag{5.4}$$

The corresponding probabilities for a successful transit through the *i*-th sub-network and the successful response of the manageable devices are given by:

$$\begin{aligned} s_{net}(i) &:= 1 - f_{net}(i) \qquad , i = 1,..,SN \\ s_{device} &:= 1 - f_{device} \end{aligned} \tag{5.5}$$

The successful transit through all sub-networks is given by the product of all sub-network probabilities $s_{net}(i)$. A status request reaches a manageable device when all sub-networks are successfully passed. Thus, s_{net_all} and f_{net_all} can be defined as follows:

$$\begin{aligned} s_{net\_all} &:= \prod_{n=1}^{SN} s_{net}(n) \\ f_{net\_all} &:= 1 - s_{net\_all} \end{aligned} \tag{5.6}$$

The expected number of requests issued by the management station is the sum of the initial request added by the probability of further requests due to network or device failures within *R* retries. Packet losses can occur during the way through the network, after the successful transit to the destination at the device, and finally, on the way back through the network of a successfully delivered request. This failure conditions are expressed in the following formula:

$$\overline{REQ} := 1 + \sum_{r=1}^{R} \left(f_{net\_all} + s_{net\_all} \cdot f_{device} + s_{net\_all} \cdot s_{device} \cdot f_{net\_all} \right)^r \tag{5.7}$$

The multiplication of the expected number of requests $\overline{REQ}$ with the number of manageable devices *D,* results in the expected total number of requests from the NMS to all devices, considering network and device failures. With the expected total number of requests the distribution within the sub-networks can be calculated. To calculate the average number of messages within each sub-network only successfully passing messages have to be considered. The expected number of messages in each sub-network is expressed by the functions $NET_{request}(i)$ and $NET_{response}(i)$ with *i=1..SN,* distinguishing between request and response messages.

The number of request messages that reach a sub-network can be specified by the conditional success probabilities of each previous sub-network multiplied by the expected total number of requests from the NMS. The calculation for a response message is similar, with only considering the messages which are successfully delivered to the manageable devices which respond.

$$NET_{request}(i) := \begin{cases} \overline{REQ} & , \text{i} = 1 \\ \overline{REQ} \cdot \prod_{n=1}^{i-1} s_{net}(n) & , 1 < i \le SN \end{cases}$$
$$NET_{response}(j) := \begin{cases} \overline{REQ} \cdot s_{net\_all} \cdot s_{device} & , \text{j} = \text{SN} \\ \overline{REQ} \cdot s_{net\_all} \cdot s_{device} \cdot \prod_{n=j+1}^{SN} s_{net}(n) & , 1 \le j < SN \end{cases} \quad (5.8)$$

Note that the probabilities of successfully delivered messages in each sub-network (response) have to be calculated in reverse order. The figure 5-4 gives an example with 3 sub-networks and clarifies the functions in (5.8) further.

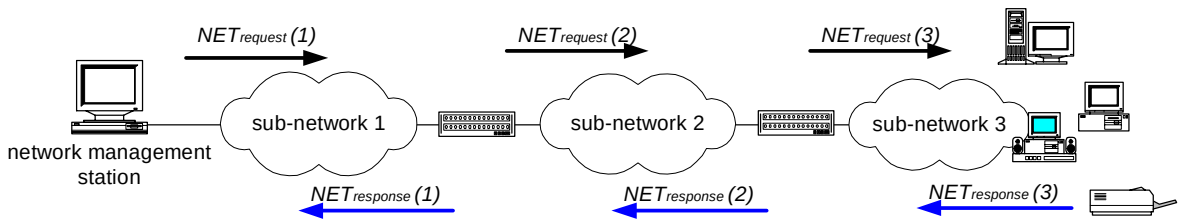

Figure 5-4: Example Network with 3 Sub-Networks

The expected management traffic ($V_{request}$ and $V_{response}$) in each *i*-th sub-network can be determined in bytes by multiplying them by the number of devices *D* and the packet sizes $PS_{request}$ for requests and $PS_{response}$ for responses. The expected total traffic V_{SNMP} is given by the sum of the terms.

$$V_{request}(i) := D \cdot NET_{request}(i) \cdot PS_{reqest} \quad ,i = 1,..,SN$$
$$V_{response}(i) := D \cdot NET_{response}(i) \cdot PS_{response} \quad ,i = 1,..,SN$$
$$V_{SNMP} := \sum_{i=1}^{SN} \left(V_{request}(i) + V_{response}(i) \right) \quad (5.9)$$

For a network without failures in sub-networks and in manageable devices we get the same formula as in (5.3) with $V'_{SNMP} = V_{SNMP}$. In chapter 5.2.3, this model will be applied to a specific network and the results of this model are compared with the other network management approaches. First however, a similar model for a mobile agent based management is discussed.

5.2.2 Network Management based on Mobile Agents

The main difference between network management based on mobile agents and SNMP is the use of the *Transmission Control Protocol* (TCP, [RFC793]) by mobile agents. Although the Agent Transfer Protocol (ATP) can be kept simple, TCP turns out to be very complex so that simplified assumptions have to be made in the first step of modeling. For instance, the *sliding window* and *slow start* mechanisms have not been considered.

TCP subdivides the data stream into several *Maximum Transport Units* (MTU). For a typical local area network (e.g. Ethernet), the size is assumed to be 1500 bytes. The header size consists of 20 bytes for IP and 20 bytes for TCP, ignoring the optional header fields. Thus, each MTU transports a maximum of 1460 bytes of user data. Similar to the previous SNMP calculation, the expected number of MTUs (message packets) is calculated. For the calculation of the expected traffic amount the differentiation between control packets and the user data packets, containing the mobile agent is important.

In this simplified version all control and user data packets are acknowledged. Finally, the number of MTUs needed to establish a TCP connection and to close this connection is added to the total number of MTUs. In TCP a 3-way handshake (2 control messages from the sender 1 acknowledge from the receiver) during set-up is needed, which is realized with the minimum MTU size of 40 bytes. Additionally, 4 MTUs are necessary for disconnection, which results in 7 MTUs altogether. The total number of MTUs without failure probabilities for a mobile agent can be calculated as following:

$$
\begin{aligned}
&MA := \text{size of a mobile agent including ATP overhead} \\
&MTU_{MA} := \left\lceil \frac{MA}{1460} \right\rceil \\
&MTU_{request} := MTU_{MA} + 4 \\
&MTU_{response} := MTU_{MA} + 3 \\
&MTU_{TCP} := MTU_{request} + MTU_{response} = 2 \cdot MTU_{MA} + 7
\end{aligned}
\tag{5.10}
$$

The definitions for network and device failure (5.4), (5.5), and (5.6) for the sub-networks remain for this calculation. The expected number of MTUs delivered by the sender can be calculated taking advantage of the formula (5.7).

$$
\overline{MTU_{request}} := \overline{REQ} \cdot MTU_{request} \tag{5.11}
$$

The term considers the number of MTUs required for a agent transmission and MTUs re-transmitted because of network or device failures. The number of acknowledgements which belong to MTUs that successfully reached the device have to be calculated by means of the formula in

(5.8). Note, that the above given formula is used to calculate the expected data traffic in the network and does not express the probability of a successful migration of a mobile agent.

The expected number of bytes transferred by the MTUs have to be calculated, distinguishing between the protocol overhead (multiplication of the MTUs with their minimum size, i.e. 40 bytes), the mobile agent data, and acknowledges.

$$
\begin{aligned}
V^{MA}_{protocol}(i) &:= 4 \cdot NET_{request}(i) \cdot 40 && ,i = 1,..,SN \\
V^{MA}_{request}(i) &:= MTU_{MA} \cdot NET_{request}(i) \cdot 1500 && ,i = 1,..,SN \\
V^{MA}_{response}(i) &:= \left(3 + MTU_{MA}\right) \cdot NET_{response}(i) \cdot 40 && ,i = 1,..,SN \\
V_{MA} &:= \sum_{i=1}^{SN} \left(V^{MA}_{protocol}(i) + V^{MA}_{request}(i) + V^{MA}_{response}(i)\right)
\end{aligned}
\qquad (5.12)
$$

For the sake of simplicity, the formula does not distinguish MTUs which are smaller than 1500 bytes. V_{MA} is the expected traffic for the migration of one mobile agent through the whole network. The total amount of expected data traffic in the network with D manageable devices can be calculated by multiplying the number of devices to V_{MA}.

5.2.3 Comparison of Network Management Strategies

In the previous section, an analytical model has been introduced, which can be used to calculate the average network traffic. Additionally, simulations of the different network management approaches have been implemented to verify the model. Detailed aspects about the simulation can be taken from the thesis [Met98] that has been worked out during the JAE project. To keep the model simple, only the local area network of the department of RWTH has been regarded that is managed by one NMS. The scenario comprises three sub-networks: the network from the NMS to the *Gateway* (GW) of the computer science, the sub-network from the gateway to the LAN of the department, and the local area network itself (see figure 5-2). Based on UDP measurements of the RWTH network we have assumed network failures of 10% in the backbone and the computer center network (NMS-GW) and 1% in both the computer science network (GW-LAN) and the department local network (LAN). The total number of nodes is 105, including printers and switches. It has been predicted that personal computers are switched off during night and weekends so that a reasonable value for devices being switched off is 30%.

The calculated numbers shown in figure 5-5 depict the expected data traffic at each sub-network. In comparison to the mobile agent approach, the expected SNMP data traffic at the NMS-GW network is higher because all packages have to be transmitted through the whole network on network or device failures. The mobile agent approach causes less traffic on the way to the LAN, but consumes too much network resources within the LAN. Each migration

of the agent means additional network load by means of code and data, which is in no relation to the information retrieved from the devices. Based on programming experiences with the agent system 5 KBytes have been assumed for the size of the mobile agent, resulting in the expected total traffic of 694 KBytes. In comparison to the expected total SNMP traffic of 79 KBytes this is not a corresponding result.

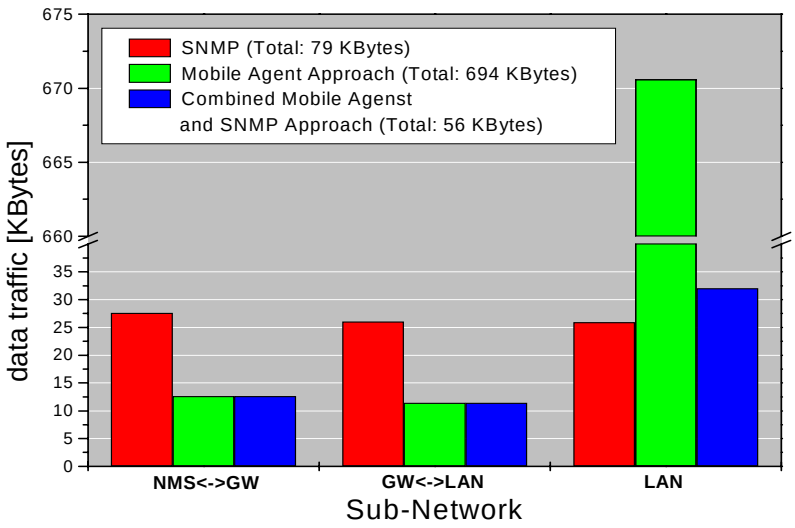

Figure 5-5: Comparison of Network Management Strategies

The calculation of the combined strategy of mobile agents polling manageable devices in the local area network seems to yield better results. The transport of the mobile agent to the LAN causes less traffic than SNMP; on the other hand, the expected data traffic in the LAN is higher because the size of the mobile agent is added to the SNMP traffic. This result makes clear that the size of the mobile agent plays an important role and the expected data traffic of 56 KBytes has to be regarded carefully.

In general, the combined strategy can be seen as an alternative to the SNMP based management, offering the advantages of the remote-programming paradigm. The beneficial effects of this combined strategy become even more obvious when more than one local network is considered. Thinking of various mobile agent sizes and higher numbers of manageable sub-networks, it can be stated that network management applications is an important and valuable field for agent technology.

5.3 Conclusion

In this chapter, a brief overview of network management approaches has been given with a discussion on drawbacks and positive characteristics of the most commonly used management protocols. The main focus was on the integration of mobile agent technology into different network management designs and their modelling. The analytical model proposed allows a direct comparison of the expected amount of data traffic in different sub-networks, thus showing the strengths and the weaknesses of these architectures. The results of the mathematical models stress the advantages of an architecture combining the traditional client/server approach with mobile agent technology; i.e. the most efficient management approach is to use mobile agents on network level and requests (of these agents) on sub-network level. This results in a decentralized NMS with mobile agents moving to selected locations and polling end-devices in the corresponding sub-network. Furthermore, dynamic updating of management applications is easily possible with the mobile agent paradigm, and the autonomous work of agents allows a more distributed control of all network devices. Even adding and withdrawing sub-networks turns out to be a zero-effort administration work. It makes sense to use SNMP for polling to continuously support legacy applications and devices. The analytical model has to be improved further because TCP has been simplified in this approach. The integration of the complete TCP overhead into the mathematical model seems to be necessary to get a more realistic evaluation and to achieve even more precise results.

Chapter 6

Conclusion and Perspectives

Mobile agent technology is a young and challenging research field and its exhaustive emergence emphasizes that mobile agent technology discloses one of the most important and interesting computing paradigms since the object oriented design and client/server-based distributed systems. The advantages mobile agents offer are the unique opportunities for structuring and implementing distributed systems. The network infrastructure for mobile agents overcome the heterogeneous environment and offer a high degree of flexibility and efficiency to the user by performing much of the work somewhere in the network. The demand for distributed applications realizing value added services grows and solutions for them are highly requested, e.g. personalized and scalable mobile services. In particular, service provider and mobile operator have an emerging interest in scalability, easily personalized service usage, and optimal distribution of their service application. An important aspect in this context is the explosive development of wireless and cellular networks, the still growing popularity of Internet, and the extreme success in miniaturization of mobile devices. The change of the *World Wide Web* into a *service network* and mobile operator's efforts in mobile portals lead to the demand of wireless access to services and are a fundamental motivation for this thesis. The ability of autonomous and disconnected operations of mobile agents makes them a high-potential technology for the 3rd generation mobile communication networks such as the *Universal Mobile Telecommunication System*. However, along with the potential benefits, autonomy and disconnected operations of mobile agents in wireless environment raise fundamental problems, which so far have found little attention in existing mobile agent systems.

The foundation of this thesis is given by the architecture of the service-based agent system and the infrastructure of the agent network - the *Java Agent Environment - JAE*. JAE is introduced as a framework for developing and deploying mobile agents and agent-based services. Adhering to standards, the core technology for mobility, services, security, management, communication, and thus, implementation of user created agents are discussed. The distinguishing features of JAE are the distributed trading facility, the involvement of standardized directory serv-

ices and adjusted agent protocols. The second goal of the thesis was to develop and implement concepts to support mobile computing. Therefore, strong attention is paid to the management of mobile agents at the boundary to wireless access networks. The tracking of mobile agents in general and the handling of mobile agents, which are unable to reach their destination and their support in case of failures have been subject to this research. Finally, the mathematical modelling of an agent-based application rounds off this thesis. The analytical study of an envisaged network management scenario stresses the beneficial application of the agent technology.

The thesis establishes a common starting-point by first introducing the mobile agent technology and defining the terminology and the components of a mobile agent system in general. The property of the paradigm and the concepts, which come with this technology are explained in detail. The status quo is given by the recommendations and proposals of the standardisation body FIPA and a survey of existing agent systems. Using this elementary knowledge, the fundamental work of the thesis is described - JAE and its agent system. The overview of the JAE architecture is given by a top-down approach. First, the Java based mobile agent middleware is introduced, then a detailed description of the components of the agent system is provided. The core technology for agent mobility, agent security, agent services and agent communication is discussed in detail with hierarchical module representation and class structure as well as process diagrams. The service trading facilities introduced by the so called *service center* with implementation details and programming examples to mediate the *service-based agent programming* paradigm has been proven to be the right concept by several measurements with different service trader approaches and a varying number of requests to the service center. The combination of this new service trading and the integration of the Agent Directory by means of the *Agent Directory Protocol* (ADP) introduce the *plug-and-participate* concept of the JAE agent systems. Remaining at agent protocols, another result is that both examined communication concepts - the blackboard communication based on the Linda architecture and the message passing communication - turned out to be important and complementary concepts to be supported by the *Agent Communication Protocol* (ACP).

The agent migration is a key to mobile agents and therefore, much effort concerning security, persistency, and efficiency has been put into the research of the *Agent Transport Protocol* (ATP). The consideration of mobile devices and thus, the support of mobile computing has lead to the *Ticket* concept introducing ATP manipulation, security contemplation, and mobile agents' resource requirements. Examination of implementation details show the interplay of the ATP, the persistence storage, and the protocol handler and clarify this complex subject. Practical tests and measurements underline protocol security and code persistency, but at the same time make clear the loss of performance in the agent migration. Based on these results, investigations on cellular and wireless local area networks have been made to put focus on the second research topic - the support of services in mobile computing. On the basis of the *Global System for Mobile Communication* (GSM), the problems with mobile agents and the requirements of the agent network and the agent system at the boarder to wireless access networks are

described. An obvious solution to management of disconnected and malfunctioning mobile agents is demonstrated by the so-called *kindergarten* and *maintenance* concepts, which are both realized with the service-based agent programming paradigm of JAE. Summarizing, the service center, the kindergarten service, and the maintenance service enable a tracking and handling of mobile agents both during their scheduled execution, and in the presence of unforeseen errors and failures. Additional tools have been tested, such as the *invoker service*, which allows the involvement of very small mobile devices with only limited capacities, or the *launcher* which offers an API to embed agent support in any Java application so that these programs can be simply extended and gain from the mobile agent technology. Concluding, one can say that all these concepts provide crucial support for an agent-based approach to mobile computing, which has been demonstrated with prototype implementations within the JAE testbed. It has shown that mobile agents are predestined for personal disconnected operations - especially in wireless access networks. The study of the mobility support ends with a reflection of the agent technology in the *Telecommunication Information Networking Architecture* (TINA) and shows a possible solution based on JAE.

As an example for the final analytical study of an agent-based application, the network management has been selected. The main focus was on the integration of mobile agent technology into different network management designs and their modelling. The analytical model proposed allows a direct comparison of the amount of data traffic in different sub-networks (including wireless access networks), thus showing the strengths and the weaknesses of these architectures. The results of the mathematical models stress the advantages of an architecture combining the traditional client/server approach with mobile agent technology. This results in a decentralized *Network Management System* (NMS) with mobile agents moving to selected locations and polling end-devices in the corresponding sub-network. The analytical model has to be further improved because TCP has been simplified in this approach. The integration of the complete TCP overhead into the mathematical model seems to be necessary to get a more realistic evaluation and to achieve more precise results.

Major experiences gained during the implementation and testing phase of this thesis include the easy realization of agent-based distributed applications and the flexibility that the mobile agent concept offers to application programmers. Once the set of service agents has been defined and established, the combination of these services to fulfil tasks by means of mobile agents is simple. The service-based agent programming concept - introducing the different agent classes with system and mobile agents - shows the advantage of JAE’s architecture. Although, none of the distributed software applications and solutions are bound to mobile agent technology, it has shown that mobile agents are valuable *design patterns* for distributed software development. The agent technology comes along with a set of convenient tools and libraries allowing very fast designing and prototyping of distributed applications. Advantages in fast application development and prototyping cannot be described in numbers and benchmarks, it is more a practical experience that programmers have to experiment with and under-

stand. Summarizing, there are no specific “mobile agent applications”, but there are plenty of application areas that highly benefit from using mobile agent technology.

Already today, companies such as the Apollis interactive AG uses successfully a subset of the agent technology for mobile mass market services, e.g. *Location Based Services* (LBS) for mobile phones [PLW02]. LBS allows new and for end customers convenient value added mobile services, e.g. agents take over the task autonomously tracking and monitoring users and delivering information right in time - customized, and at the appropriate location [Par02]. System agents fulfil interface jobs outstandingly well and are the first choice in system integration modules because of their autonomous work and their scalability. Notwithstanding these facts, the intensive standardization efforts and research of the *Foundation of Intelligent and Physical Agents* (FIPA) in the agent technology confirm the necessity of further exhaustive research.

Obviously, many unforeseen and new value added services can result from the agent technology, the Internet, and the telecommunication services, which remain a challenging work to investigate. Due to the problems of open environments and security issues, it is to be expected that agent technology will first emerge in closed environments. Even so, my understanding for mobile agent applications is that once a mobile device, depending on the location - whether at home or somewhere else in the world - will be loaded via a wireless access network with a mobile agent that assists to control and manage all electronic devices and communication facilities in the surroundings that the owner is allowed to and want to use.

Appendix A

Glossary

A

AC	Authentication Center	85
ACC	Agent Communication Channel	25
ACL	Agent Communication Language	8, 16, 23, 76
ACP	Agent Communication Protocol	34, 77
ACTS	Advanced Communications Technologies and Services	20, 82
AD	Agent Directory	17, 96
AI	Artificial Intelligence	1, 119
AMASE	Agent-based Mobile Access To Multimedia Information Services	112
AMP	Agent Meeting Point	94
AMS	Agent Management System	25
AP	Agent Platform	25
API	Application Programming Interface	22, 34, 91
ARPA	Defense Advanced Research Projects Agency	69
ARQ	Automatic Repeat Request	86
ASE	Application Service Elements	114
ATM	Asynchronous Transfer Mode	104
ATP	Agent Transfer Protocol	16, 31, 98, 126

B

BSC	Base Station Controller	85
BSS	Base Station Subsystem	85
BTS	Base Transceiver Station	85
B-WiN	Breitband-Wissenschaftsnetz	104

C

CDMA	Code-Division Multiple Access	81

CI Communication Interface 21
CLIMATE Cluster for Intelligent Mobile Agents for Telecommunication Environments .. 20
CMIP Common Management Information Protocol 114
CMIS Common Management Information Service 114
CMISE Common Management Information Service Element 114
CORBA Common Object Request Broker Architecture 15, 109
CSCW Computer Supported Cooperative Work 8
CSMA/CA Carrier Sense Multiple Access with Collision Avoidance 84

D

DAI Distributed Artificial Intelligence 8
DARPA Defense Advanced Research Projects Agency 8
DCOM Distributed Component Object Model 21
DCS Digital Cellular System 81
DECT Digital Enhanced Cordless Telephony 84
DF Directory Facilitator 25
DHCP Dynamic Host Configuration Protocol 91
DNS Domain Name System 91
DPE Distributed Processing Environment 109
DSSS Direct Sequence Spread Spectrum 84

E

EDGE Enhanced Data Rates for GSM Evolution 82
EIR Equipment Identity Register 85
ETSI European Telecommunications Standards Institute 79

F

FEC Forward Error Correction 86
FIPA Foundation for Intelligent Physical Agents 3, 8, 21, 23

G

GGSN Gateway GSN 88
GMSC Gateway MSC 85
GPRS General Packet Radio Service 80
GPS Global Positioning System 92
GSM Global System for Mobile Communication 3, 80
GSN GPRS Support Nodes 88
GUI Graphical User Interface 34
GUID Global Unique IDentifier 74
GW Gateway 127

H

HLR Home Location Register 85
HTTP Hypertext Transport Protocol 93, 103

I

IDL Interface Definition Language 10
IETF Internet Engineering Task Force 115
IIOP Internet Inter Object Request Broker Protocol 27
IMEI International Mobile Equipment Identity 85
IMT2000 International Mobile Telecommunication 2000 79
IN Intelligent Network 20
IPMT Internal Platform Message Transport 27
ISC International Switching Center 85
ISDN Integrated Services Digital Network 85
ISO International Organization for Standardization 114
ISP Internet Service Provider 88

J

J2ME Java 2 Micro Edition 66
JAE Java Agent Environment 3, 5, 31
JAR Java Archive 62
JDK Java Development Kit 36
JMS Java Message Service 70
JVM Java Virtual Machine 16, 32

K

kbps kilobit per second 82
KIF Knowledge Interchange Format 8
KQML Knowledge Query and Manipulation Language 8

L

LAN Local Area Network 104
LBS Location Based Services 134
LD Location Directory 90
LDAP Lightweight Directory Access Protocol 43
Lisp LISt Processing language 5
LMT Local Maintenance Terminal 85
LMU Ludwig-Maximilian Universität München 104

M

MAC Medium Access Protocol 84
MARS Mobile Agent Reactive Spaces 70
MASE Mobile Application Support Environment 91
MASIF Mobile Agent System Interoperability Facility 3, 21
Mbps Megabit per second 83
MIB Management Information Base 114
MID Mobile Information Device 66
MIDP Mobile Information Device Profile 66
MSC Mobile Switching Center 85
MTU Maximum Transport Units 126

N

NBS Narrow Band Socket 80
NMA Network Management Agent 115
NMS Network Management Station 115
NSS Network-Subsystem 85

O

ODP Open Dirstributed Processing 40
OMC-R Operation and Maintenance Center Radio 85
OMG Object Management Group 17, 21
OMS Operation and Maintenance Subsystem 85
ORB Object Request Broker 11, 53
OSI Open Systems Interconnection 114
OTA Over The Air 67

P

PA Provider Agents 110
PCN Personal Communications Network 82
PCS Personal Communications Services 81
PDA Personal Digital Assistant 31, 79
PDN Packet Data Network 88
PDU Packet Data Unit 88
PDU Protocol Data Unit 114
PLMN Public Land Mobile Network 85
PSTN Public Switched Telephone Network 85

Q

QoS Quality Of Service 19, 92

R

RAP Region Access Points 22
RLP Radio Link Protocol 86
RMI Remote Method Invocation 21, 40
RPC Remote Procedure Call 9
RSS Radio-Subsystem 85
RWTH Rheinisch-Westfälische Technische Hochschule 3, 31

S

SGSN Serving GSN 88
SIM Subscriber Identification Module 85
SLAPD Standalone LDAP Access Protocol Daemon 42
SMG Special Mobile Group 82
SMS Short Message Service 80
SMSC SMS Center 87
SMTP Simple Mail Transport Protocol 103
SNMP Simple Network Management Protocol 40, 113
SOAP Simple Object Access Protocol 73
SOS Save Our Soul/Ship 101
SS7 Signalling System 7 85
SSL Secure Socket Layer 67
SSM Service Session Manager 110

T

TCL Tool Command Language 5
TIA Telecommunication Industry Association 82
TINA Telecommunication Information Networking Architecture 108
TINA-C TINA-Consortium 108
TMN Telecommunications Management Network 20

U

UA User Agents 110
UAL UMTS Adaptation Layer 92
UCP Universal Computer Protocol 105
UDP User Datagram Protocol 115
UML Unified Modelling Language 24, 64
UMTS Universal Mobile Telecommunication System 79
USM User Service Session Manager 110

V

VHE Virtual Home Environment 108

VLR Visited Location Register 85

W

WAN Wide Area Networks 104
WAP Wireless Application Protocol 79, 103
WLAN Wireless Local Area Network 84
WWW World Wide Web 2

X

XML Extensible Markup Language 73

Appendix B

List of Figures, Programs, and Tables

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Appendix C

Bibliography

[AbPe99] C. Abarca, M. Peters. *TINA Business Model for UMTS: Benefits and Possible Enhancements*. Proceedings of TINA '99, Ohahu, April 1999.

[ACG86] S. Ahuja, N. Carriero, D. Gelernter. *LINDA and Friends*. IEEE Computer, 19(8):26-34, August 1986.

[ADK02] ADK. *Agent Development Kit*. Tryllian Solutions B.V., <http://www.tryllian.com/>, Version 2.1, November 2002.

[AMA99] ACTS AC346. *AMASE - Agent-based Mobile Access To Multimedia Information Services*, <http://www.cordis.lu/infowin/acts/rus/projects/ac346.htm>, January 1999.

[ArLa98] Y. Aridor, D.B. Lange. *Agent Design Patterns: Elements of Agent Application Design*. In proceedings of the 2nd International Conference on Autonomous Agents (Agents '98), ACM Press, 1998.

[BaPi98] M. Baldi, G.P. Picco. *Evaluation the Trade-offs of Mobile Code Design Paradigms in Network Management Applications*. Proceedings of the 20th International Conference on Software Engineering (ICSE '98), Kyoto, April 1998.

[BCS01] P. Bellavista, A. Corradi, C. Stefanelli. *Mobile Agent Middleware for Mobile Computing*. In: IEEE Computer, vol.34, no.3, March 2001.

[BCPR03] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa. *JADE - A White Paper.* exp, Volume 3 - n.3, <http://sharon.cselt.it/projects/jade/>, September 2003.

[BHR+98] J. Baumann, F. Hohl, K. Rothermel, M. Schwehm, M. Straßer. *Mole 3.0: A Middleware for Java-Based Mobile Software Agents*. In the proceedings of Middleware'98, Springer Verlag, 1998.

[Böl02] L. Bölöni. *The Bond 3 Agent System - A White Paper.* University of Central Florida, <http://bond.cs.ucf.edu/>, October 2002.

[CGH+95] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris. *Itinerant Agents for Mobile Computing*. IBM Research Report 20010. IBM Research Division, T.J. Watson Research Center, 1995.

[CKO+99] D. Carrega, G. Kolonias, S. Oldeboershuis, V. Patrascu, H. Rahoui, B. Schiemann, D. Trifanescu. *Delivering Value-Added Banking Services for Mobile Customers with the AMASE Agent Environment*. Proceedings of the ACTS Mobile Summit, Sorrento, 1999.

[Cla98] Claption et. al. *TINA Concepts for Third Generation Mobile Systems - TINA concepts applied to UMTS and concepts for mobility support in TINA*. EURESCOM Project P608, deliverable 2, <http://www.eurescom.de/~pub-deliverables/P600-series/P608/D2/>, January 1998.

[CLDC02] CLDC. *The CLDC HotSpot Implementation Virtual Machine. Java 2 Platform, Micro Edition (J2ME). White Paper.* Revised Version 1.1. <http://java.sun.com/products/cldc/>, Sun Microsystems, Inc., July 2002.

[CLI98] T. Magedanz (editor). *Agent Cluster Baseline Document, Version 0.2*. <http://www.fokus.gmd.de/research/cc/ima/climate/doc/baseline-doc.html>, February, 1998.

[CLZ00] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli. *MARS: A Programmable Coordination Architecture for Mobile Agents*. IEEE Internet Computing, Vol. 4, No. 4, pp. 26-35, July/August 2000.

[CMV+00] F. Chatzipapadopoulos, T. Magedanz, I. Veneris, G. de Zen, F. Zizza. *Harmonised Internet and PSTN service provisioning.* In Special Issue on Mobile Agents for Telecommunication Applications, Computer Communications Journal, Elsevier Publishers, Vol. 23, No.8, April 2000.

[Con97] Concordia. *Concordia: An Infrastructure for Collaborating Mobile Agents*. Mitsubishi Electric Information Technology Center America, Horizon Systems Lab., 1997.

[COR02] CORBA. *The Common Object Request Broker Architecture: Core Specification*. Version 3.0.2, Object Management Group, Inc., December 2002.

[DCS96] I-ETS 300 020-3. *European digital cellular telecommunications system (Phase 1). Mobile station conformance test system, Part 3: DCS 1 800 mobile station conformity specification*. Edition 2, ETSI, December 1996.

[DECT03] EN 300 175-1. *Digital Enhanced Cordless Telecommunications (DECT); Common Interface (CI); Part 1: Overview.* Version 1.7.1, ETSI, July, 2003.

[DGNS98] E. Dahlman, B. Gudmundson, M. Nilsson, J. Sköld. *UMTS/IMT-2000 Based on Wideband CDMA*. IEEE Communications Magazine, Vol. 36 No. 9, September 1998.

[Dou98] R.E. Douglas. *SNEAKERS: A Concurrent Engineering Demonstration System*. Master thesis of Worcester Polytechnic Institute, October 1998.

[DPRS03] EN 301 649. *Digital Enhanced Cordless Telecommunications (DECT); DECT Packet Radio Service (DPRS).* Version 1.3.1, pp. 15-22 (4.1 Data Service Structure), ETSI, March 2003.

[EbVö99] J. Eberspächer, H.J. Vögel. *GSM - Switching, Services and Protocols*. John Wiley & Sons Ltd., Chichester, 1999.

[EHLR80] L.D. Erman, F. Hayes-Roth, V.R. Lesser, D.R. Reddy. *The HEARSAY II speech-understanding system: Integrating knowledge to resolve uncertainty.* Computing Surveys, 12(2), 1980.

[ElBi99] M. El-Darieby, A. Bieszczad. *Intelligent Mobile Agents: Towards Network Fault Management Automation*. In Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management, pp. 611-622, May 1999.

[Emm98] M. Emmerich. *Implementierung und Bewertung einer Dienstvermittlung für Mobile Agenten*. Diploma Thesis, Prof. Dr. Spaniol, RWTH Aachen, 1998.

[ErLe75] L. Ermann, V. Lesser: *A Multilevel Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge*. In Fourth International Joint Conferences on Artificial Intelligence, September 1975.

[ERM97] ETS 300 133-3. *Electromagnetic compatibility and Radio spectrum Matters (ERM);. Enhanced Radio Message System (ERMES) Part 3: Network Aspects.* Edition 2, pp. 124-132 (9 Network interworking), ETSI, November 1997.

[ETSI] ETSI. *European Telecommunications Standards Institute*. <http://www.etsi.org/>, 1988-2003, last updated August 2003.

[Fas98] A. Fasbender. *Messung und Modellierung der Dienstgüte paktevermittelder Netze*. Aachener Beiträge zur Informatik, Band 24, Aachen, 1998.

[FFMM94] T.W. Finin, R. Fritzson, D. McKay, R. McEntire. *KQML As An Agent Communication Language*. In proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), pp. 456-463, Gaithersburg, Maryland, December 1994.

[FIPA02] FIPA. *FIPA Specification 2002*. Foundation for Intelligent Physical Agents, <http://www.fipa.org/repository/bysubject.html>, 2002.

[FKK96] A.O. Freier, P. Karlton, P.C: Kocher. *The SSL Protocol, Version 3.0*. Netscape Communications, <http://wp.netscape.com/eng/ssl3/>, November 1996.

[FMMO99] A. Furuskär, S. Mazur, F. Müller, H. Olofsson. *EDGE: Enhanced Data Rates for GSM and TDMA/136 Evolution.* IEEE Personal Communications Magazine, Vol.6 No. 3, June 1999.

[FRG+99] A. Fasbender, F. Reichert, E. Geulen, J. Hjelm, T. Wierlemann. *Any Network, Any Terminal, Anywhere*. IEEE Personal Communications Magazine, Vol.6, No. 2, April 1999.

[FrGr96] S. Franklin, A. Graesser: *Is it an agent or just a program?: A taxonomy for autonomous agents*. In proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages. <http://www.msci.memphis.edu/~franklin/AgentProg.html>, Springer-Verlag, 1996.

[Fün98] S. Fünfrocken. *Transparent Migration of Java-Based Mobile Agents: Capturing and Reestablishing the State of Java Programs*. Kurt Rothermel and Fritz Hohl (Eds.), Mobile Agents. LNCS 1477, Sepetember, 1998.

[GCK+02] R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson, D. Rus. *D'Agents: Applications and Performance of a Mobile-Agent System.* John Wiley & Sons, Software: Practice and Experience, 32 (6), pp. 543-573, May 2002.

[GeFi92] M.R. Genesereth, R.E. Fikes at al. *Knowledge Interchange Format, Version 3.0 Reference Manual*. Logic Group Technical Report Logic-92-1. Interlingua Working Group of the DARPA Knowledge Sharing Effort, Stanford Logic Group, June 1992.

[GhVi97] C. Ghezzi, G. Vigna. *Mobile Code Paradigms and Technologies: A Case Study.* K. Rothermel and R. Popescu-Zeletin (Eds.) Mobile Agents, LNCS 1219, April 1997.

[GIOP02] GIOP. *Common Object Request Broker Architecture: Core Specification*. Version 3.0.2., pp. 15.1 - 15.64 (Chapter 15, GIOP), Object Management Group, Inc., December 2002.

[Gol96] G. Goldszmidt. *Distributed Management by Delegation*. Ph.D. Thesis, Columbia University, 1996.

[GoYe98] G. Goldszmidt, Y. Yemini. *Delegated Agents for Network Management*. IEEE Communications: Management of Heterogeneous Networks, Vol.36, No.3, March 1998.

[GPRS01] TS 101 356. *Digital cellular telecommunications system (Phase 2+). General Packet Radio Service (GPRS). Mobile Station (MS) supporting GPRS*. Version 7.2.0. ETSI, April 2001.

[GPRS02] TS 101 351. *Digital cellular telecommunications system (Phase 2+). General Packet Radio Service (GPRS). Mobile Station - Serving GPRS Support Node (MS-SGSN), Logical Link Control (LLC) layer specification*. Version 8.7.0, ETSI, January 2002.

[GPRS03] TS 101 348. *Digital cellular telecommunications system (Phase 2+). General Packet Radio Service (GPRS). Interworking between the Public Land Mobile Network (PLMN) supporting GPRS and Packet Data Networks (PDN)*. Version 7.10.0. ETSI, July 2003.

[GSMA] GSMA. *Global System for Mobile Communications Association - the world wide web site*. <http://www.gsmworld.com/>, 1999-2003, last updated August 2003.

[HAK03] H. Harroud, M. Ahmed, A. Karmouch. *Policy-Driven Personalized Multimedia Services for Mobile Users*. IEEE Transactions on Mobile Computing (Vol. 2, No. 1, pp. 16-24), IEEE Computer Society, January-March 2003.

[Har96] J. Harmer. *The OnTheMove Project*. In the proceedings of the 3rd International Workshop on Mobile Multimedia Communications (MoMuC-3), Princeton New Jersey, September 1996.

[HKB97] F. Hohl, P. Klar, J. Baumann. *Efficient Code Migration for Modular Mobile Agents*. 3rd ECOOP Workshop on Mobile Object Systems: Operating System support for Mobile Object Systems, 1997.

[ICM98] IEEE Communications Magazine. *ACTS Mobile Programm Europe*. IEEE Communications Society, Vol.36 No. 2, pp.80-136, February 1998.

[IEEE98] *Mobile Software Agents for Telecommunications*. IEEE Communications, Vol. 36, No. 7, July 1998.

[IKV03] IKV++ Technologies AG. *Grasshopper - The Agent Platform 2*. <http://www.grasshopper.de/>, Version 2.2, January 2003.

[IPCM95] IEEE Personal Communications Magazine. *Special Issue - The European Path Toward UMTS*. IEEE Communications Society, Vol.2 No. 1, February 1995.

[IPCM97] IEEE Personal Communications Magazine. *IMT-2000: Standards Efforts of the ITU*. IEEE Communications Society, Vol. 4 No. 4, August 1997.

[IPCM98] IEEE Personal Communications Magazine. *Paving the Way to Third Generation Mobile Systems in Europe*. IEEE Communications Society, Vol. 2 No. 2, April 1998.

[ITU] ITU. *International Telecommunication Union*. <http://www.itu.int/>, last updated August 2003.

[Java03] Java. *The Java Tutorial - A practical guide for programmers*. SUN Microsystems Inc., <http://java.sun.com/docs/books/tutorial/>, May 2003.

[JDK12] SUN Micro System Inc. *Java 2 SDK, Standard Edition Documentation*. Version 1.2.2. Java Software Division, <http://java.sun.com/products/jdk/1.2/docs/>.

[JGA+96] P.C. Janca, D. Gilbert, M. Aparicio, B. Atkinson, S. Pritko, J. Rusnak. *Practical Design of Intelligent Agent Systems*. IBM Corporation, Research Triangle Park, North Carolina, June, 1996.

[JS03] SUN Microsystems Inc. *JavaSpaces Service Specification*. Jini Network Technology. Version 2.0. <http://wwws.sun.com/software/jini/specs/js2_0.pdf>, June 2003.

[KGN+97] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, G. Cybenko. *Agent TCL: Targeting The Needs Of Mobile Computers*. IEEE Internet Computing, July/August, 1997.

[KGN+99] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, G. Cybenko. *Mobile Agents for Mobile Computing*. In Dejan S. Milojicic and Frederick Douglis and Richard G. Wheeler, editors, Mobility: Processes, Computers, and Agents, chapter 14.3, Addison Wesley and ACM Press, April 1999.

[KiNo03] P. Kim, Y. Noh. *Mobile Agent System Architecture for Supporting Mobile Market Application Service in Mobile Computing Environment*. In the proceedings of the International Conference on Geometric Modeling and Graphics (GMAG'03), London, July 2003.

[Kle75] L. Kleinrock. *Queueing Systems, Volume 1: Theory.* John Wiley & Sons, pp. 89-114., 1975.

[KPM+98] B. Kreller, A.S. Park, J. Meggers, G. Forsgren, E. Kovacs, M. Rosinus. *UMTS: A Middleware and Mobile-API Approach*. IEEE Personal Communications Magazine, Vol. 5 No. 2, April 1998.

[Küp01] A. Küpper. *Nomadic Communication in Converging Networks*. Dissertation, RWTH Aachen, VDE Verlag GmbH, Berlin, January 2001.

[KüPa98] A. Küpper, A.S. Park. *Stationary vs. Mobile User Agents in Future Mobile Telecommunication Networks*. Kurt Rothermel and Fritz Hohl (eds.): Mobile Agents, LNCS 1477, 1998.

[KüPa99] A. Küpper, A.S. Park. *Realizing Nomadic Communication with Mobile Agents: Strategies and their Evaluation*. Proceedings of TINA '99, Ohahu, April 1999.

[LaOs98] D.B. Lange and M. Oshima. *Mobile Agents with Java: The Aglet API*. <http://aglets.sourceforge.net/>. World Wide Web Journal, 1998.

[LCX+01] T.J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, P. Bowman. *Hitting the distributed computing sweet spot with TSpaces*. Computer Networks 35, pp. 457-472, ELSEVIER, 2001.

[LiDr95] A. Lingnau, O. Drobnik. *An Infrastructure For Mobile Agents: Requirements And Architecture*. Fachbereich Informatik (Telematik), Johann Wolfgang Goethe-Universität Frankfurt am Main, 1995

[Lip00] S. Lipperts. *On the Efficient Deployment of Mobility in Distributed System Management*. In: 3rd International Workshop "Mobility in Databases & Distributed Systems", Greenwich, UK, September 2000.

[LiPa98] S. Lipperts, A.S. Park. *Managing CORBA with Agents*. Proceedings of the 4th International Symposium on Interworking, Ottawa, July 1998.

[LiPa99] S. Lipperts, A.S. Park. *An agent-based middleware – a solution for terminal and user mobility.* "Mobile Agents in Intelligent Networks and Mobile Communication Systems" in the Journal Computer Networks 31, ELSEVIER, 1999.

[Lin98] C. Linnhoff-Popien. *CORBA, Kommunikation und Management*. Springer Verlag, Berlin, Heidelberg, New York, 1998.

[LISP] Association of Lisp Users (ALU). *LISt Processing language*. <http://www.lisp.org/>, 1997-2003, last updated August 2003.

[LiYe99] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification. Second Edition*. <http://java.sun.com/docs/books/vmspec/>, Sun Microsystems, Inc., 1999.

[LNPS97] Lombardo, P. Nicosia, S. Palazzo, M. Samaratto. *Service Architecture Support of Personal Mobility in a Multi-Domain Environment.* Proceedings of TINA '97 - Global Convergence of Telecommunications and Distributed Object Computing, pp. 51-58, IEEE Computer Society, Santiago, November 1997.

[Lud00] R.E. Ludwig. *Eliminating Inefficient Cross-Layer Interactions.* Dissertation, RWTH Aachen, <http://iceberg.cs.berkeley.edu/papers/Ludwig-Diss00/>, April 2000.

[LZCH02] J. Liu, S. Zhang, J. Cao, J. Hu. *An Agent Enhanced Framework to Support Predispatching of Tasks in Workflow Management Systems*. In the Proceedings of Engineering and Deployment of Cooperative Information Systems: First International Conference, Beijing, LNCS 2480, pp. 80-90, Springer-Verlag, Heidelberg, September 2002.

[Mae95] P. Maes. *Artificial Life Meets Entertainment. Life like Autonomous Agents.* Communications of the ACM, 38, 11, pp. 108-114, 1995.

[MaGl02] T. Magedanz, R. Glitho. *Mobile Agents in Telecommunications*. IEEE Network Magazine, Vol. 16, No. 3, May 2002.

[MAL00] MAL - *The Mobile Agent List*, <http://mole.informatik.uni-stuttgart.de/mal/mal.html>, March 2000.

[MaPZ96] T. Magedanz, R. Popescu-Zeletin. *Intelligent Networks - Basic Technology, Standards and Evolution*. International Thomson Computer Press, London, June 1996.

[MASIF98] OMG. *MASIF Specification 98*. OMG TC Document, <http://www.fokus.fraunhofer.de/research/cc/ecco/masif/>, March 1998.

[Mat96] B. Mathiske. *Mobilität in persistenten Objektsystemen*. Dissertation, Department of Computer Science, University of Hamburg, 1996.

[Met98] P. Metternich. *Bewertung eines agentenbasierten Netzwerkmanagements im Vergleich zu SNMP.* Diploma thesis of the Department of Computer Science, Prof. Dr. Spaniol, RWTH Aachen, October 1998.

[MIDP02] MIDP. *Mobile Information Device Profile 2.0*. JSRs: Java Specification Requests, JSR 118. Java Community Process Expert Group, <http://jcp.org/en/jsr/all>, November 2002.

[MiRa00] P. Misikangas and K. Raatikainen. *Agent Migration Between Incompatible Platforms*. In the proceedings or the 20th International Conference on Distributed Computing Systems (ICDCS 2000), April 2000.

[MLZ03] M. Mamei, L. Leonardi, F. Zambonelli. *Co-Fields: A Unifying Approach to Swarm Intelligence.* 3rd International Workshop on Engineering Societies in the Agents World (ESAW) 2002. Madrid, Spain. Lecture Notes in Computer Science, No 2577, Springer Verlag, pp. 68-81, 2003.

[MMU01] N.H. Minsky, Y. Minsky, V. Ungureanu. *Safe Tuplespace-based Coordination in Multiagent Systems*. Applied Artificial Intelligence 15(1): pp. 11-33, January 2001.

[MPF+98] J. Meggers, A.S. Park, A. Fasbender, B. Kreller, L. Alonso. *A Multimedia Communication Architecture for Handheld Devices*. In the proceedings of the PIMRC 98, Boston, September 1998.

[MPL96] J. Meggers, A.S. Park, R. Ludwig. *Roaming between GSM and wireless LAN*. ACTS Mobile Communication Summit, Granada, November 1996.

[MRK96] T. Magedanz, K. Rothermel, S. Krause. *Intelligent Agents: An emerging technology for the next generation telecommunications?* In proceedings of the INFOCOM'96, San Francisco, March 1996.

[MSP97] J. Meggers, T. Strang, A.S. Park. *A Video Gateway to Support Video Streaming to Mobile Clients*. In the proceedings of the ACTS Mobile Communication Summit 97, Aalborg, October 1997.

[NBS97] Intel and Nokia. *Narrow Band Socket Specification*, Rev.1.0. March 1997.

[Nes98] S. Neskakis. *Bewertung von Kommunikationsformen in heterogenen Mobile Agenten-Systemen*. Diploma thesis of the Department of Computer Science, Diploma thesis of the Department of Computer Science, Prof. Dr. Spaniol, RWTH Aachen, September 1998.

[OjPr98] T. Ojanperä, R. Prasad. *An Overview of Air Interface Multiple Access for IMT-2000/UMTS*. IEEE Communications Magazine, Vol. 36 No. 9, September 1998.

[OMG] Object Management Group. *OMG - Object Management Group*.<http://www.omg.org/>, 1997-2003, last updated August 2003.

[OMG00] OMG. *Agent Technology, Green Paper*. Agent Working Group. OMG Document ec/2000-08-01, version 1.0, August 2000.

[OTM98] ACTS AC034. *OTM - On The Move ACTS project*. <http://www.cordis.lu/infowin/acts/rus/projects/ac034.htm>, September 1998.

[Ous98] J. K. Ousterhout. *Scripting: Higher Level Programming for the 21st Century.* IEEE Computer Magazine, March 1998.

[Ox74] *Oxford Advanced Learner's Dictionary of Current English.* Oxford University Press 1974.

[Paging01] ETSI TR 123 908. *Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); Technical Report on Pre-paging*. 3GPP TR 23.908 version 3.0.1, ETSI, August 2001.

[PaLe97] A.S. Park, S. Leuker: *A Multi-Agent Architecture Supporting Services Access*. K. Rothermel and R. Popescu-Zeletin (Eds.) Mobile Agents, LNCS 1219 (pp. 62-73), April 1997.

[PaLi98] A.S. Park, S. Lipperts. *Prototype Approaches to a Mobile Agent Service Trader.* 4th International Symposium on Interworking “Interoperability of networks for interoperable services”, Interworking98, Ottawa, Canada, July 1998.

[PaMe97] A.S. Park, J. Meggers. *Mobile Middleware: Additional functionalities to cover wireless terminals*. David J. Goodman and D. Raychaudhuri (Eds.), Mobile Multimedia Communications, Plenum Publishing (pp. 151-157), New York, 1997.

[Par02] A.S. Park. *Location Based Services - präzise Mehrwertinformationen für den jeweiligen Aufenthaltsort*. Published in Computer Zeitung, 23. September 2002.

[PaRe98] A.S. Park, P. Reichl: *Personal Disconnected Operations with Mobile Agents*. 3rd Workshop on Personal Wireless Communications, PWC'98, Tokyo, April 1998.

[Pei02] H. Peine. *Application and Programming Experience with the Ara Mobile Agent System*. Software Journal - Practice and Experience, 2002.

[Per98] C.E. Perkins. *Mobile IP - Design Principles and Practices*. Addison-Wesley, Massachusettes, 1998.

[PES98] A.S Park, M. Emmerich, D. Swertz. *Service Trading for Mobile Agents with LDAP as Service Directory.* IEEE 7th Intl. Workshop on Enabling Technologies “Infrastructure for Collaborative Enterprises”, WETICE'98, Stanford University, California, USA, June 1998.

[Pet97] J.C. Petrie. *What's an agent, and what's so intelligent about it?*. IEEE Internet Computing, July/August, 1997.

[PiBu02] G.P. Picco, M. L. Buschini. *Exploiting Transiently Shared Tuple Spaces for Location Transparent Code Mobility.* Farhad Arbab, Carolyn L. Talcott (Eds.): Coordination Models and Languages, 5th International Conference, York, Springer LNCS 2315, pp. 258-273, April 2002.

[PKL97] A.S. Park, A. Küpper, S. Leuker. *Jae - A Multi-Agent System with Internet Services Access.* Forth International Conference on Intelligence in Services and Networks “Technology for Cooperative Competition”, Como. Al Mullery et al. (Eds.), LNCS 1238, pp. 155-164, May 1997.

[PLKS99] A.S. Park, S. Lipperts, B. Kreller, B. Schiemann. *Mobility Support with a Mobile Agent System*. 3rd International Workshop on Intelligent Agents for Telecommunication Applications, Stockholm, August 1999.

[PLW02] A.S. Park, S. Lipperts, Marc Wilhelm. *Location Based Services for Context Awareness - Moving from GSM to UMTS*. In the proceedings of the SSGRR Conference, Italia, January 2002.

[PMM89] A. Patel, G. McDermatt, C. Mulvilill. *Integrated Network Management and Artificial Intelligence*. Integrated Network Management (I), North-Holland, 1989.

[PMT+97] A.S. Park, J. Meggers, V. Travnicek, G. Forsgren, E. Kovacs, M. Rosinus. *Mobile Application Support Environment - MASE*. 4th International Workshop on Mobile Multimedia Communications (MoMuC97), Seoul, October 1997.

[PSS02] T.R. Payne, R. Singh, K. Sycara. *Calendar Agents on the Semantic Web*. IEEE Computer Society, IEEE Intelligent Systems, Vol. 17(3), pp. 84-86, May/June 2002.

[PSW96] C. Popien, G. Schürmann, K.-H. Weiß. *Verteilte Verarbeitung in Offenen Systemen*. B.G. Teubner Verlag, Stuttgart 1996.

[PWW00] B. Pagurek, Y. Wang, T. White. *Integration of Mobile Agents with SNMP: Why and How.* IEEE/IFIP Network Operations and Management Symposium, Honolulu, April 2000.

[Ram98] L. Raman. *OSI System and Network Management*. IEEE Communications: Management of Heterogeneous Networks. Vol.36, No. 3, March 1998.

[Rap95] J. Rapeli. *UMTS: Targets, System Concepts, and Standardization in a Global Framework*. IEEE Personal Communications Magazine, Vol.2 No. 1, February 1995.

[RBM00] M. Ranganathan, M. Bednarek, D. Montgomery. *A Reliable Message Delivery Protocol for Mobile Agents*. In: Proceedings of 2nd International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents (ASA/MA), Zurich, Switzerland, September 2000.

[RFC768] J. Postel. *User Datagram Protocol ()*. DARPA Networking Group Report RFC 768, August 1980.

[RFC791] J. Postel. *Internet Protocol (IP)*. DARPA Networking Group Report RFC 791, September 1981.

[RFC793] J. Postel. *Transmission Control Protocol (TCP)*. DARPA Networking Group Report RFC 793, September 1981.

[RFC1034] P.V. Mockapetris. *Domain Names - Concepts and Facilities*. RFC 1034, November 1987.

[RFC1035] P.V. Mockapetris. *Domain Names - Implementation and Specification*. RFC 1035, November 1987.

[RFC1065] K. McCloghrie, M.T. Rose. *Structure and identification of management information for TCP/IP-based internets.* RFC 1065, August 1988.

[RFC2002] C.E. Perkins, ed. *IPv4 Mobility Support*. RFC 2002, October 1996.

[RFC2131] R. Droms. *Dynamic Host Configuration Protocol*. RFC 2131, March 1997.

[RFC2132] S. Alexander, R. Droms. *DHCP Options and BOOTP Vendor Extensions*. RFC 2132, March 1997.

[RFC2251] M. Wahl, T. Howes, S. Kille. *Lightweight Directory Access Protocol (v3)*. RFC 2251, December 1997.

[Rhe99] A.J. Rhem. *The Role of Intelligent Agents in the Information Infrastructure*. A.J. Rhem & Associates, Inc., 1999.

[RhMa00] B.J. Rhodes and P. Maes. *Just-in-time information retrieval agents*. IBM Systems Journal, Vol. 39, Nos. 3&4, pp. 685-704, 2000.

[Rie94] D. Riecken. *M: An Architecture of Integrated Agents*. Communications of the ACM, Vol.37, No.7, pp. 107-113, July, 1994.

[RMI03] SUN Microsystems Inc. *Java Remote Method Invocation*. Release 1.4.2. Java Software Division, <http://java.sun.com/products/jdk/rmi/>, April 2003.

[RoDr03] G. van Rossum and F.L. Drake, Jr. *Python Tutorial*. PythonLabs, <http://www.python.org>, Release 2.3, July 2003.

[RPC97] RPC. *CDE 1.1: Remote Procedure Call*. CAE Specification, document number C706, The Open Group, <http://www.opengroup.org/>, October 1997.

[RuNo95] S. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach,* Prentice Hall, 1995.

[Sat01] I. Satoh. *Adaptive Protocols for Agent Migration*. In the proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS'2001), pp.711-714, IEEE Computer Society, April 2001.

[SLA96] University of Michigan: *The SLAPD and SLURPD Administrator's Guide*. <http://www.umich.edu/~rsug/ldapdoc/guides/sldap/1.html>, Release 3.3, April 1996.

[SM00] Nokia Mobile Phones Ltd. *Smart Messaging Specification*. Revision 3.0.0, December 2000.

[SMS02] TS 100 901. *Digital cellular telecommunications system (Phase 2+). Technical realization of the Short Message Service (SMS), Point-to-Point (PP).* Version 7.5.0, ETSI, February 2002.

[SNB+00] D. O'Sullivan, J. Núñez-Suárez, H. Brouchoud, P. Cros, C. Moore, C. Byrne. *Experiences in the use of FIPA agent technologies for the development of a Personal Travel Application*. In the proceedings of 4th International Conference on Autonomous Agents, June 2000.

[SOAP00] W3C. *Simple Object Access Protocol (SOAP) 1.1*. W3C Technical Reports and Publications <http://www.w3.org/TR/SOAP/>, May 2000.

[SPM94] O. Spaniol, C. Popien, B. Meyer. *Dienste und Dienstvermittlung in Client/Server-Systemen*. International Thomson Publication, 1994.

[SSF00] P. Simões, L. Moura e Silva, F. Boavida Fernandes. *A Generic Management Model for Mobile Agent Infrastructures.* Proceedings of SCI 2000 – The 4th World Multiconference on Systemics, Cybernetics and Informatics, Florida, USA, July 2000.

[SSS+99] L. Silva, P. Simoes, G. Soares, et al. *JAMES: A Platform of Mobile Agents for the Management of Telecommunication Networks*. In: Proceedings of the 180 Bibliography Third International Workshop on Intelligent Agents for Telecommunication Applications, Stockholm, Sweden, August 1999.

[Sta98] W. Stallings. *SNMP and SNMPv2: The Infrastructure for Network Management*. IEEE Communications: Management of Heterogeneous Networks. Vol.36, No. 3, March 1998.

[Swe98] D. Swertz: *Analyse von Sicherheitsaspekten in Mobile-Agenten Systemen*. Diploma Thesis, RWTH Aachen, December 1998.

[TCL03] TCL. *Tool Command Language*. <http://www.tcl.tk/>, Release 8.4.4, July 2003.

[ThTh97] L.L. Thomsen, B. Thomsen. *Mobile Agents - The new paradigm in computing*. ICL System Journal, May 1997.

[TINA97] L. Kristiansen (Ed.): *Service Architecture - Version 5.0*, TINA-C Deliverable, <http://www.tinac.com/>, June 1997.

[TWH90] V. Tschammer, A.Wolisz, J.Hall. *Support for Cooperation and Coherence in an open Service Environment.* In Proceedings Second IEEE Workshop on Future Trends of Distributed Computing System, pp. 222-228, 1990.

[UMTS] UMTS Forum. *Universal Mobile Telecommunications System Forum*, <http://www.umts-forum.org>, last updated August 2003.

[UM3004] TR 101 397 V3.0.1. *Universal Mobile Telecommunications System (UMTS). Concept groups for the definition of the UMTS Terrestrial Radio Access (UTRA).* UMTS 30.04 version 3.0.1. ETSI, October 1998.

[VHE00] TR 122 970 V3.0.1: *Universal Mobile Telecommunications System (UMTS); Service aspects; Virtual Home Environment (VHE).* 3G TR 22.970 version 3.0.1, ETSI, January 2000.

[VKK01] G. Valetto, G. Kaiser, G.S. Kc. *A Mobile Agent Approach to Process-Based Dynamic Adaptation of Complex Software Systems.* In the proceedings of Software Process Technology: 8th European Workshop, EWSPT 2001. Lecture Notes in Computer Science Volume 2077, pp. 102-117, Springer-Verlag Heidelberg, June 2001.

[Voy03] Voyager. *Voyager Messaging Developer's Guide.* <http://www.recursionsw.com/products/voyager/voyager.asp>, Release 4.5, Recursion Software, Inc., January 2003.

[Wag98] G. Wagner. *Conceptual Foundations of Artificial Agents.* Habilitation thesis of the University Leipzig, pp. 1-11. December 1998.

[WAP02] WAP. *Wireless Application Protocol, WAP 2.0, Technical White Paper.* Open Mobile Alliance, Ltd., <http://www.wapforum.org>, January 2002.

[WDP00] WDP. *Wireless Datagram Protocol*. Version 1.2.1, WAP Forum, <http://www.wapforum.org/>, 2000.

[Whe03] Thomas Wheeler. *Developing Peer Applications with Voyager.* Recursion Software, Inc., <http://www.recursionsw.com/products/whitepapers/whitepapers.asp>, May 2003.

[Whi94] J.E. White. *Telescript Technology: The Foundation for the Electronic Marketplace*. White Paper, General Magic, 1994.

[Whi96] J.E. White. *Telescript Technology: Mobile Agents.* General Magic White Paper, 1996.

[WLAN99] ANSI/IEEE Std 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. Institute of Electrical and Electronics Engineers Inc. Edition, 1999.

[WLR98] B.F.C. Wind, F. Lucidid, P. Reynolds. *Open Service Architecture for Mobile and Fixed Environments (OSAM)*. Final Release, ACTS Ref. AC036 DOLMEN, <http://www.fub.it/dolmen/>, August 1998.

[WPB99] T. White, B. Pagurek, A. Bieszczad. *Network Modeling For Management Applications Using Intelligent Mobile Agents*. Journal of Network and Systems Management, Special Issue on Mobile Agent-based Network and Service Management, September 1999.

[WPD02] T. White, B. Pagurek, D. Deugo. *Management of Mobile Agent Systems: Learning from the Ants*. Hamid R. Arabnia (Ed.): Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '02, Las Vegas, USA, Volume 4. CSREA Press, pp. 1831-1837. June 2002.

[X.400] F.400/X.400. *Message handling system and service overview.* ITU-T Recommendation F.400/X.400, June 1999.

[X.500] X.500. *Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services*. ITU-T Recommendation X.500, February 2001.

[X.700] ISO - *International Organization for Standardization. Information Technology – Open Systems Interconnection – Management Framework for Open Systems Interconnection*. CCITT Recommendation X.700 – ISO/IEC 7498-4, 1992.

[X.950] X.950. *Open distributed processing - Trading function: Specification*. ITU-T Recommendation, August 1997.

[XML03] W3C. *Extensible Markup Language (XML)*. W3C Architecture Domain. <http://www.w3.org/XML/>, June 2003.

Curriculum Vitae

Anthony Sang-Bum Park

Date and place of birth	December 17, 1968 in Seoul (South Korea)
1975 - 1979	Städtisch Katholische Grundschule Köln (Primary School)
1979 - 1988	Dreikönigsgymnasium Köln (High School)
19.05.1988	Abitur (School-leaving Exam and University Entrance Qualification)
1988 - 1990	University of Koblenz-Landau
14.11.1990	Vordiplom at the University of Koblenz-Landau (Bachelor of Computer Science)
1990 - 1995	Aachen University, RWTH
12.07.1995	Diploma of Computer Science at RWTH (Master Degree)
1995 - 2000	Ph.D. scientist at Aachen University, Department of Computer Science, Informatik 4
2000-2003	Founder and member of the board of Apollis interactive AG