Integrating MASIF and FIPA Standards for Agent and Agent System Interoperability

Sven Kaffille
Distributed Systems Group
Otto Friedrich Universität Bamberg
Feldkirchenstrasse 21
96052 Bamberg, Germany
email: sven.kaffille@wiai.uni-bamberg.de

Guido Wirtz
Distributed Systems Group
Otto Friedrich Universität Bamberg
Feldkirchenstrasse 21
96052 Bamberg, Germany
email: guido.wirtz@wiai.uni-bamberg.de

ABSTRACT
In order to use agent technology for implementing complex software systems, widely available software platforms are as crucial as methods to support their interoperability. The paper discusses the concepts of a system that combines the benefits of the two most accepted standards in the agent world, namely FIPA and MASIF, into a platform that permits the interaction as well as migration of agents from both worlds. Additionally, an implementation based on the FIPA-OS implementation of the FIPA standard is described.

KEY WORDS
software agents, platforms, interoperability, standards

1 Introduction

In the last decades the complexity of tasks that are carried out by software as well as the complexity of the software's environment, e.g. due to distributed

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

systems, has grown rapidly. Efforts to establish new technologies for modelling and implementing software in order to handle this kind of complexity are manyfold. Among them, software agents are regarded as the next step of software technology [1]. Agent technology promises to make the modelling and development of heterogenous and distributed systems easier, as it provides a higher abstraction level than other approaches. In order to make use of the benefits of agent technology beyond the early development steps of analysis and design, platforms that provide services for agents and on which agents can operate are required for implementing agent-based designs directly into agent systems.

To faciliate agent technologie's diffusion open standards for these platforms are necessary to enable interoperability between agents and agent platforms of different vendors. For interoperability common interfaces of agents and agent systems are required, that are independent of the agents' technical environment. This aspect has been addressed by the *Foundation for Intelligent Physical Agents* (http://www.fipa.org/) by its so-called FIPA-standards. The focus of FIPA standards is on interoperability of agents and multi-agent systems through standard interfaces on the level of agents' communication and a common agent management reference model. As a second requirement, standard interfaces of agents are required for interoperability of different programmers' agents and standard interfaces of agent systems are recommended to ease the administration of agent systems of different vendors. This aspect is handled by the *Object Management Group* (OMG) through the *Mobile Agent System Interoperability Facility* (MASIF) standard for mobile agent systems based on OMG's CORBA [16]. The standard focusses on interoperability on the agent system level and security mechanisms for mobile agents. Agent to agent communication has not been specified by the OMG. Agent management, agent transfer, agent and agent system names as well as agent system type and location syntax have been standardized.

As the FIPA standard specifies interoperability on the agent communication level and MASIF addresses minimum interoperability requirements on the agent system level it seems promising to combine both standards into one agent platform. This paper shows how this can be done by extending an existing FIPA compliant agent platform - FIPA-OS - to be MASIF compliant. For this purpose, the following problems have to be solved in a way that is consistent with both platforms:

1. Naming of agents, agent platforms and systems,
2. integration of different infrastructures,
3. implementing agent mobility, and

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

4. applying appropriate security mechanisms.

Due to the state of the art of the used platform regarding security, interoperability w.r.t. this issue has to be left to future work. For the extension of FIPA-OS the goal is to reuse as many existing components of FIPA-OS as possible and to keep the extension decoupled from the original FIPA-OS in order to permit easy migration to new versions of the platform. In order to evaluate the proposed solutions to the problems stated above, the concepts have been implemented into an interoperable platform for both standards.
The rest of the paper is organized as follows: In section 2, a short description of the FIPA standard and the MASIF standard is given as this forms the basis of our work. Section 3 discusses how these standards can be integrated addressing the problems 1 – 4 step by step. Section 4 describes an implementation of the concepts based on FIPA-OS. Section 5 points out related work whereas the concluding section 6 summarizes the results and addresses some important open issues.

2 Agent Standards

For completeness reasons this section describes the achievements of the FIPA in multi-agent system standards, the first implementation of these standards – FIPA-OS – and the OMG's results in standardizing multi-agent systems. While FIPA focusses on interoperability of agents and multi-agent systems through standard interfaces on the level of agent communication, the MASIF standard focusses on interoperability on the agent system level. Agent to agent communication has not been specified by the OMG.

2.1 The FIPA Standard and FIPA-OS

FIPA (http://www.fipa.org/) was founded in 1996 with the goal to produce standards for heterogenous and interoperable Agents and Multi-Agent Systems. The specifications of the FIPA Standard are build through input and collaboration of FIPA's membership that currently consists of about 60 members. Since 1996 FIPA has produced over 90 specifications and about 25 of them have been approved by FIPA's members to be standards. A FIPA compliant *Agent Platform* (AP) provides the physical infrastructure to deploy agents and implements the FIPA Agent Management Reference Model [2]. According to this reference model an AP consists of one or more host computers, their operating systems, agent support software, agents that are executed on the host computers and some agent management components. These components are:

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

- The *Agent Management System* (AMS) is a mandatory component, has supervisory control over the AP and provides White Pages Services.
- The *Directory Facilitator* (DF) provides Yellow Pages Services (optional).
- The *Message Transport Service* (MTS) is the default communication service between agents on different APs and on the same AP. It is designed to ensure interoperability between agents on FIPA compliant APs. It is provided by a so called *Agent Communication Channel* (ACC) [3].

The internal design of an AP is not specified by FIPA and left to the developers of FIPA compliant APs.

Every FIPA agent is identified by an unique *Agent Identifier* (AID) which consists of the agent's name, a list of communication addresses (an URL) and a list of resolution services for the agent's addresses. FIPA suggests that an agent's name should have the form `agentname@APname` where the AMS of the agent's AP occurs in the latter.

The AMS and DF provide interfaces to de/register, search and modify entries in their directories. In addition, the AMS provides a function to obtain the description of an AP. Every agent created on or moved to an AP is registered with the AMS of this AP by it's AID. Agents providing services to other agents may be registered with one or more DFs by their names and service descriptions. DFs can be searched for agents by service descriptions. For convenience of agents and their programmers DFs can be federated with other DFs so that an agent's search is automatically extended to other DFs.

The standard interfaces of AMS and DF are accessible through the FIPA *Agent Communication Language* (ACL), which is the most important outcome of FIPA's work. The ACL has a layered architecture. Every message expressed in ACL has a so-called *Communicative Act* (CA) based on speech act theory [5] as its type [4]. Every ACL-Message carries a number of parameters as sender and receiver which are filled with values for each instance of a message [6]. The content of a message is formulated in a *Content Language* (CL) (e.g. [7]) and the symbols used with that CL are defined by an *ontology*. The content and the ontology are also parameters of an ACL message. Another parameter defines which *Interaction Protocol* (IP) (e.g. [8]) the message belongs to. IPs are often used and standardized patterns of message sequences. FIPA maintains a library of 11 IPs. Additionally the flow of messages can be tracked by a conversation identifier. FIPA specified standard representations of its ACL (e.g. in XML [9]). For the transport between agents (on the same or different APs) ACL messages are encapsulated

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

in envelopes. The representation of an envelope is also standardized in [10], [11]. The messages are sent by the ACC of an AP which can use one of three *Message Transport Protocols* (MTPs) to send ACL messages to a remote agent on another FIPA compliant AP. These MTPs are HTTP [12], IIOP [13] and WAP [14]. The MTP for message transport between agents on the same AP is – as other internal design matters – not defined by FIPA. Every FIPA compliant AP has to implement at least one standard MTP, one standard envelope representation and one standard ACL representation. These specifications ensure interoperability between FIPA compliant APs if the assumptions made above are fulfilled. For interoperability between agents they have to implement mechanisms to deal with the same CLs and ontologies.

The most important implementation of the FIPA specifications has been carried out by FIPA-Open Source (FIPA-OS) that is developed and maintained by emorphia (http://www.emorphia.com/), one of FIPA's members (FIPA-OS is implemented in Java and currently available in version 2.2.0 under an open source license from http://fipa-os.sourceforge.net/). One can implement own agents by extending a class (`FIPAOSAgent`) called the agent shell [15]. With every agent a *Conversation Manager* (CM), a *Task Manager* (TM) and a set of tasks are associated. The CM keeps track of the agent's conversations with other agents. Every agent can execute tasks in parallel. These tasks are managed and executed by the TM. From within tasks ACL-messages can be sent and received. Therefore tasks are associated with one or more conversations the ACL messages belong to. The TM makes sure that the incoming ACL messages received by the CM are delivered to the correct task(s) via callback methods.

The ACL messages are sent via an MTS that implements a CORBA and a HTTP MTP for communication with agents residing on remote platforms. As MTP for AP internal communication Java Remote Method Invocation (RMI) is used. There is no support for mobile agents in FIPA-OS. In FIPA specifications mobility of agents is only considered for the agent life cycle [2] and in a deprecated specification that suggests an integration with OMG's *Mobile Agent System Interoperability Facility* (MASIF) standard.

2.2 The OMG MASIF-Standard for Mobile Agent Systems

The OMG MASIF standard for mobile agent systems [16] is based on OMGs famous Common Object Request Broker Architecture (CORBA). Agent management, agent transfer, agent and agent system names as well as agent system type and location syntax have been standardized by the OMG. For this

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

purpose agents are divided into *stationary* and *mobile agents*. Agent Systems provide an execution environment for these agents and the services to transfer mobile agents. Agents and agent systems are associated with an *Authority*. Agents reside in execution environments of agent systems called *place*s. An agent system of one authority is situated with other agent systems of the same authority in a *region* that is regarded as a security domain. Agents and agent systems of different regions can be interconnected via agent systems – called *Region Access Point* – of these regions that are exposed to the outside world. Every region owns a naming service or shares one with another region. Agent systems are interconnected via a *Communication Infrastructure* that is implemented using CORBA.

With `MAFFinder` and `MAFAgentSystem`, two standard CORBA interfaces and data structures for agent systems have been specified. `MAFFinder` has to be implemented by a region's naming service and provides methods for finding, de/registering agents, places and agent systems. The interface `MAFAgentSystem` is implemented by every MASIF compliant agent system and provides methods for creation, transfer and querying information about the agent system.

MASIF specifies how an agent's state and code can travel from one agent system to another using methods of the `MAFAgentSystem` interface. A mobile agent is associated with a *Codebase* from where his code can be loaded. A mobile agent requests its migration from the agent system it currently resides on via an internal API. For this purpose the destination agent system and place are announced by the agent. Then the agent is suspended, the pieces of the agent's state to be transferred are identified, the agent code and state are serialized and encoded for the chosen transport protocol by the agent system. Afterwards the agent is transferred. The destination agent system has to determine whether it can interpret the agent or not and, dependent on that decision, accept the agent. After acceptance, the agent is decoded and deserialized, instantiated, its state is restored and its execution can resume. Agent transfer may only take place between agent systems of the same agent system type, as every agent platform provides a different execution environment. Different strategies to transfer an agent's code are considered:

1. Automatic transfer of all possible classes.
2. Automatic transfer of the agent class only, other classes are transferred on demand.
3. Combinations of the first and second strategy.

Agent systems can maintain a cache of agents' code so that the code needs not to be transferred every time an agent is transferred.

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

One important additional concern with mobile agents is *security* which is addressed by MASIF, too. When agents are transfered or created from remote clients, the clients and the involved agent systems are mutually authenticated automatically. When an agent from one agent system communicates with an agent on another agent system both agent systems try to authenticate the remote agent and remote agent system to apply the appropriate security policies. Agents and agent systems have to be able to control access to their services based on other agents' and agent systems' permissions According to the MASIF specification the security needs are met by a CORBA compliant secure ORB with Security Level 2. Not all ORBs support this security level.

3 Integration of FIPA and MASIF

Information provided on the agent communication level by FIPA is also provided on the agent system level by MASIF, e.g. agent lookup, agent registering etc. In order to integrate the standards, the concepts of FIPA have to be mapped onto the concepts of MASIF and the other way round. For this purpose the issues raised in section 1 are discussed and solved step by step.

1. Place, agent and AP naming
2. Mapping of infrastructure of MASIF onto FIPA's infrastructure and the other way round.
3. How to implement agent transfer.
4. Security concerns.

3.1 Naming Issues

Place, agent and AP names have to be unique to identify agents. This must also be true when agents are transferred from their home AP to another AP.

For places here the namingscheme `placename-place@ap` is suggested with `placename` and `ap` replaced with the name of the corresponding place's and AP's names. If an agent is named following the agent naming scheme suggested by FIPA, uniqueness of agent names cannot be assured in the presence of agent transfer. For agents the naming scheme suggested by FIPA is applied and extended for mobile agents as follows. When an agent with name `agenta@ap1` is transferred from AP `ap1` to another AP `ap2` its name should change to `agent@ap2` as he from now executes under control of the AMS of `ap2`. If there is already an agent running on `ap2` with name `agenta@ap2`

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

there is a conflict. So an agent should be renamed by a predefined scheme when it arrives at a new AP that it is not its home AP for the first time. The scheme suggested here is for our example of `agent a` when it arrives at `ap2`: `agenta-from-ap1@ap2`. The first part of this name will not undergo any further changes when the agent arrives on other APs. In the case the agent is transferred back to its home AP its original name should be restored. In the meantime the AMS of its home AP must ensure that no further agent registers with the same name. In addition an agent being transferred from one AP to another must be forced to deregister with the AMS of the AP he moves from and to modify its service description (if any) registered with the DF as its addresses for ACL message transport change after agent transfer. An AP which is reachable via the internet is named based on the name of the host that is exposed to the internet (most likely the host where the `MAFAgentSystem` is running). This naming scheme should also be applied in LANs.

3.2 Integration of Infrastructures

The infrastructures described in MASIF and FIPA do not exactly fit to each other. The concepts of agent system and AP as well as the platform services, e.g. DF/`MAFFinder`, have to be integrated. As there is no concept of *region* in FIPA, a way that is compliant to FIPA and MASIF, has to be found to implement it. Because there is also no notion for *place* in FIPA and no equivalent in the Agent Management Reference Model, it must be thought of ways for agents to find and reason about places. This should also be specified on the ACL communication level (not only on the agent system level). Also a FIPA compliant way to register the Places with the AMS and DF must be found so that they can be looked up by agents. The concerns of the standards' communication infrastructures have to be addressed as well.

The notion of agent system in MASIF is compatible with the concept of the AP in FIPA. So the interface `MAFAgentSystem` is mapped onto the AMS that has supervisory control over an AP [17]. Regarding the DF the question about the region in MASIF and `MAFFinder` interface is raised. Theoretically the `MAFFinder` can be mapped onto a DF (as suggested in [17]), but in this case the concept of a Region is not taken into consideration as there can be more than one DF in one Region that consists of a number of APs. This can be solved by choosing one dedicated DF that is federated with all DFs in the Region and is accessible through the `MAFFinder` interface on the agent system level. As not all agents are registered with the DF the agent system has to register agents with the `MAFFinder` in order to enable `MAFFinder` to supply information about agents that provide no services and therefore are

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

not registered with the DF.

A FIPA compliant AP can comprise one or more machines so that every machine is mapped onto a place. Each Place is treated as a *special agent* that can execute other agents and provides a service-description. This description contains information about its execution environment, a description of the resources available from it and a list of agents executed within it. The description is registered with the DF of an AP and allows agents to search for, migrate to and reason about places with special resources. If an agent cannot migrate to a place, e.g. the execution environment is not compatible, it still has the option to contact an agent residing in the desired place which is able to accomplish the task to be carried out there.

To be MASIF compliant, an AP has to provide its interfaces on the agent system level via CORBA as its CI. The CI for the agents is independent from the agent system level CI namely the ACC. As there must be CORBA available on the agent system level the standard MTP of an AP is IIOP. Regarding the security mechanisms of the CI described in the MASIF standard these mechanisms should be generalized so that they are independent of an ORB implementation and could be applied to other CIs e.g. via HTTP as some FIPA APs provide an HTTP interface to the ACC and the HTTP infrastructure could also be used to transfer agents and access the interfaces on the agent system level. This could be useful in environments where no CORBA infrastructure is available.

3.3 Agent Tranfer for Mobility

There are *two principal possibilities* to implement agent transfer within the given settings:

1. Transfer agents and their code as content of ACL messages.
2. Transfer agents using low level APIs and de/serialization processes.

The *first possibility* means to transfer an agent by sending its serialized state formulated in a CL as content of an ACL message. This can be done in two ways. For both ways an universal Transfer Ontology must be standardized to facilitate transfer between APs of different vendors. Then a process how an agent's data state can be de/serialized from/to a representation in a CL with this Migration Ontology must be standardized. For the first way additionally a standardized high level interpreted Agent Programming Language (APL) implemented on many soft- and hardware platforms must be available. So the agent's script would be independent from the execution environment. Then the agent's state and script could be sent via ACL messages. The second

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

way can be applied, if no APL is available. So the sources, e.g. Java, of the agent's classes are included in an ACL message, but this would require the agent's code to be compiled at the receiving agent system.

The *second possibility* means to use APIs on the agent system level and de/serialization processes of existing programming languages, e.g. Java, to transfer agents. The first way of the first possibility would be the most interoperable way to transfer agents. The second way is as interoperable as the second possibility, because an agent needs a certain execution environment, e.g. FIPA-OS. Hence, this method would not increase the interoperability of agent transfer compared with the second possibility. In order to avoid the need to specify a migration ontology, a de/serialization process and an APL – that would *not* be a standard – for now it is sufficient to use low level APIs on the agent system level and the de/serialization mechanisms of programming languages like Java for agent transfer.

An additonal problem with agent transfer occurs when an agent migrates during ongoing conversations with other agents. Then these agents might not recognize that the agent has moved and may try to send messages to the agent's outdated address. For this reason messages belonging to conversations not finished yet are forwarded to the new location of the agent by a proxy that lives until all conversations have ended. Agents that want to communicate with a mobile agent only have to look up its addresses before they initiate a new conversation. The protocol for migration proceeds as follows. A mobile agent requests its migration from the AP it currently resides on via an internal API and anounces the destination agent system and place. The agent is suspended, deregistered with the AMS (The agent's programmer has to care of the registration with the DF) and, the pieces of the agent's state to be transferred are identified, the agent's code and state are serialized and encoded for the chosen transport protocol by the AP. If necessary a proxy is instantiated that forwards the messages of ongoing conversations or stores them until it knows the agent's new address. The agent is transferred. The destination AP has to determine if it can interpret the agent. If it can, it accepts the agent, the agent is decoded and deserialized, instantiated, its state restored, and registered with the local AMS. Afterwards execution is resumed and the agent's new address is delivered to the proxy if one was created. For the transfer of agent code the mechanisms described by MASIF can be applied.

Another matter is that an AP consists of one or more machines represented by places. Therefore agents can be transferred between different machines that constitute one and the same AP. Agent transfer can be divided into AP internal and external transfer. Internal design is left open to AP Programmers (by MASIF and FIPA) and is not an issue here. For external

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

agent transfer the CI of an agent system is used.

3.4 Security

Up to now, security mechanisms have been elaborated by FIPA only in a preliminary specification [18]. The security mechanisms described by MASIF can only be applied in CORBA environments that have a secure ORB that implements CSI level 2. They have to be elaborated so that they can also be applied in other environments. To this account interoperability regarding security issues is not solved with our current platform but has been left to future work.

4 Implementation of a FIPA and MASIF compliant Agent Platform

For the implementation of our concepts, FIPA-OS was chosen to be a good basis to be extended as it is open source, implements the latest FIPA specifications as soon as possible and has a component-oriented design. The goals for the extension are to re-use as many components of FIPA-OS as possible and to decouple our implementation of the MASIF extensions from the FIPA-OS internals in order to facilitate the combination with future FIPA-OS releases. For the latter it has been managed to implement all extensions *with no change of any line of code of the original FIPA-OS 2.2.0 release.*

4.1 Agent Transfer Infrastructure

For agents being transferred an infrastructure must be created that enables agents to move from one host to another. On every machine (place) a service that can receive and instantiate agents is run. This Receiver-Service receives an agent's code and is able to instantiate the agent and resume its execution. For internal agent transfer of an AP this service is implemented via Java RMI. For external agent transfer the `receive_agent` method of the `MAFAgentSystem` interface is used to deliver agents. An agent is instantiated (as described below) at that location if the `MAFAgentSystem` service runs on the host where the desired Place of the transferred agent is located. Otherwise the agent is delivered to the desired place via the internal agent transfer infrastructure over RMI to the Receiver-Service on the corresponding host. The Receiver-Service receives the data state of an agent serialized by the Java serialization mechanism with help of an `java.io.ObjectOutputStream`.

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

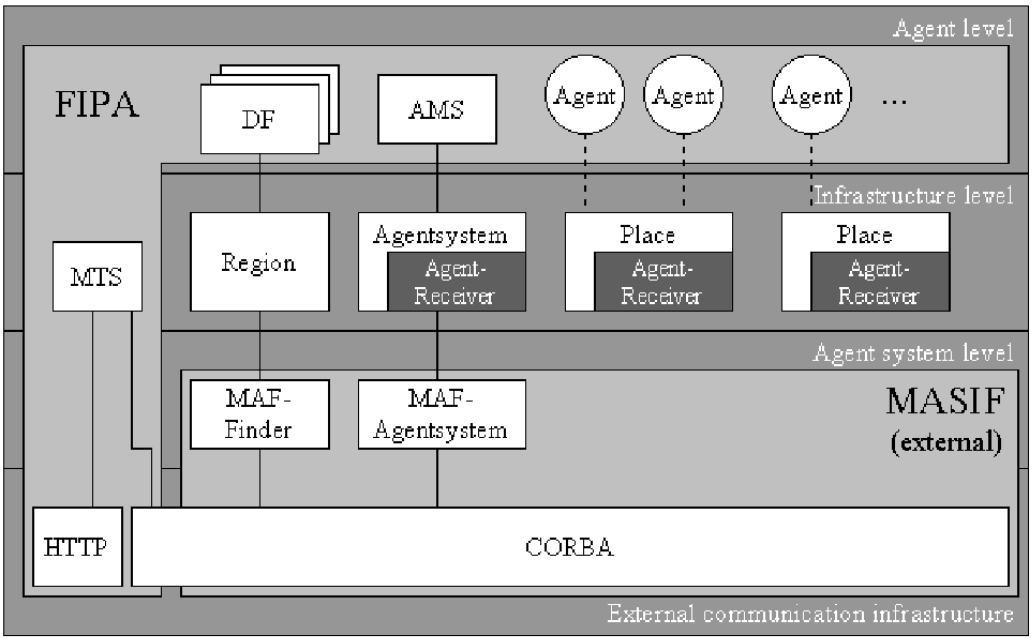

Figure 1: Overall System Architecture

When the agent is instantiated the appropriate classes must be loaded into the Java Virtual Machine (JVM). For this purpose Java uses `ClassLoader`s. As mobile agents classes may not be available locally, a `ClassLoader` for mobile agents' classes in a MASIF environment has to be able to load them from remote hosts via CORBA. Therefore a custom `AgentClassLoader` is implemented that is capable of loading classes through the `fetch_class` method of `MAFAgentSystem`. Therefore, a specific `AgentClassLoader` is provided with the address (IOR) of the codebase for the mobile agent whose transfer is currently processed. `AgentClassLoader` is used by an `AgentInputStream` class that is derived from the `java.io.ObjectInputStream` distributed with Java, but uses the `AgentClassLoader` to load classes for mobile agents.

The location from where the classes can be loaded is called codebase. The codebase of an agent can be accessed by other agent systems through the `fetch_class` method of `MAFAgentSystem`. The Codebase is distributed over the hosts of an AP, so that the classes of an agent must only be deployed to the place (that means its classes must be in the classpath) where it is instantiated the first time. As address of its codebase the `MAFAgentSystem`'s IOR is used because the codebase is accessible through it. As loading classes over a network every time an agent arrives at a place consumes bandwith each host holds a cache of agent classes. When an agent arrives the first

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

time, its classes are written to this cache.

While a JVM is running the classes are present in the cache of the JVM, but when it is shut down, the classes of mobile agents are lost. So they are written to a directory in the classpath of the JVM from where they can be loaded again. An additional issue is assuring timeliness for mobile agent classes. The code of an agent can undergo changes between two times he arrives at one place. A mechanism to avoid instantiating the agent with old classes can be based on accossiating version ids of its classes with a running agent.

The services described above are all integrated into one component `Place` that represents a place. The `Place` provides a GUI from where mobile agents and stationary agents (the original `FIPAOSAgent`) are started by the user. So the place knows about all started agents and is enabled to notify the agent aystem which agents are executed on it. This solution to register agents was choosen, as every `FIPAOSAgent` can be started in a single JVM and no mechanisms to register them with a Place have been implemented. Implementing them would require to change original FIPA-OS classes. The places register themselves with `MAFFinder`/DF via ACL-Messages when they are started up and notify it when new agents are executed. So that these information can be obtained through the methods `lookup_agent` and `lookup_place` on the agent system level and agents can find Places using ACL messages. This is done through the ACL interface of a DF following the naming scheme. With the DF a place-description (a special service-description) is registered that provides information about the execution environment present at a Place and a list of agents executed at a place.

4.2 Implementation of `MAFAgentSystem` and `MAFFinder`

The `MAFAgentSystem` interface is implemented by a component called `AgentSystem` that wraps the AMS of FIPA-OS. For this purpose this component uses the ACL and MTS/MTP components distributed with FIPA-OS to retrieve information from the AMS like lists of agents or places that can be requested through the methods `list_all_places` and `list_all_agents_of_authority`. The methods `suspend_agent` and `resume_agent` have not been implemented yet as the FIPA-OS API has no methods to suspend and resume a `FIPAOSAgent`. So not the whole agent life cycle proposed by FIPA and MASIF is implemented. To provide the codebase for an agent, `AgentSystem` is provided with a list of classes available from each Place. For this purpose the `Place`s send a list of classes to the `Agent System` when they are started up. When the `AgentSystem` is started up, the address (IOR) of the `MAFFinder` service, where the `AgentSystem` registers itself, its agents and places are sup-

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

plied. As there is no notification from the AMS to `AgentSystem` when a new agent is registered this is done transparently by the `Place` where an agent is started. When a new agent is registered with `AgentSystem`, the `AgentSystem` registers it with `MAFFinder` automatically. Hence, agents that registered no service-description with an DF still can be looked up via `MAFFinder`.

The `MAFFinder` interface is implemented by a component called `Region`. This is implemented in an analog manner to `MAFAgentSystem` and wraps the dedicated DF in the Region. When an `AgentSystem` is registered with the `Region` the dedicated DF is forced to register with the DF of the `AgentSystem` that just registered. When the method `lookup_agent` is called, the `Region` first searches the DF for the agent. For this purpose the property list from the supplied `AgentProfile` is converted to a FIPA service-description that is used to query the DF with the appropriate FIPA ACL message. If this list is empty the local information about registered agents is used.

4.3 Agent Components

The original `FIPAOSAgent` cannot be transferred, as its components - the TM, CM and its Tasks - are not serializable. Moreover, their implementation concerns that are important when an agent wants to travel are not addressed, so a specific agent that is capable of being transferred has to be implemented. For this purposes the patterns used for the original `FIPAOSAgent` are adapted for the `MobileFIPAOSAgent`. This agent is like the original one associated with a TM, CM and a set of Tasks that are transferred with their agent when it travels to another place. These serializable components are called `MobileTaskManager`, `MobileConversationManager` and `MobileTask`. The `MobileTaskManager` manages the execution of the `MobileTasks` and the `MobileConversationManager` manages the conversations the agent is involved in. An additional component – the `AgentSender` – serializes an `MobileFIPAOSAgent` and its components except itself when a transfer is requested by the agent and transfers the agent to the desired place via the appropriate protocol, i.e. RMI for internal and CORBA for external transfer.

Requests for a transfer can be made from within `MobileTask`s. The computation of tasks happens mostly in the callback methods provided by the task for received messages to be delivered to it and in the method that starts a task's execution. The tasks are executed in parallel by threads that are hold in a pool of threads. So when a transfer is requested, the TM is notified and all threads currently executing tasks are run to their ends but no new execution is started. As the threads execute the callback methods of the tasks, all tasks are halted. The `MobileConversationManager` is also notified that a transfer is in progress. A `Proxy` is instantiated and associated with the

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

MTS to receive messages for the agent that may arrive during transfer. All incoming messages are stored by this proxy. When all tasks are halted the `AgentSender` is initialised and the agent is sent through it. The transfer is initiated in a task by calling its `migrate` method. This takes the destination, an array of serializable objects and the name of a task's method (called reentry method) as parameters and has to be the last method that calls task's method, as it is the last statement that is executed in the task's method.

After the transfer was successfull the task's reentry method specified as parameter of the `migrate` method is called with the array of serializable objects. This is done by the TM after the execution of tasks has been restarted in case of a successful transfer. With the reentry method and the supplied array of serializable objects, it is possible to have a somewhat transparent transfer in abscence of full migration capabilities of the Java Runtime Environment. After succesful transfer the `AgentSender` is notified and the new address of the agent is supplied. Then the proxy is notified to send the messages arrived meanwhile to the new location of the agent. After that all remaining components of the agent including the proxy are shut down and the agent is no longer reachable via its old address.

If the transfer fails the proxy gives the messages to the `MobileConversationManager` that re-establishes its association with the MTS. The TM restarts execution and calls the method to handle failed transfers of the task that requested the transfer. This method has the same parameters as the reentry method but its name is the name of the reentry method plus "`failed`". The reentry method and its corresponding method to handle failed transfers have to be implemented by an agent's programmer. As tasks are executed in parallel a task can request a transfer while another task does. Only the first transfer request is carried out. After transfer the task whose request has not been satisfied, is notified via a method that has the same name and parameters as the reentry method. Its name consists of the reentry method's name plus "`notexecuted`". So it is left to the programmer to decide what to do in this case.

4.4 Agent Transfer Process

When a `MobileFIPAOSAgent` requests a transfer, its components are prepared for migration as described above and it is deregistered with the AMS. For external transfer the `AgentSender` serializes the agent's data state via a `java.io.ObjectOutputStream` into a array of bytes. Then it is sent via RMI to the `AgentSystem` component of the agent system, the agent's current place belongs to. From there it is sent to the agent system where the destination place is located by calling the `receive_agent` method of that agent system's

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

`MAFAgentSystem` interface. The underlying `AgentSystem` component tries to send the agent's data state via RMI to the desired place. There the agent is re-instantiated, it's name is changed according to the naming scheme described in section 3.1, it is registered with the AP's AMS, associated with the APs MTS and its execution is resumed. To reinstantiate the agent the `AgentClassLoader` is used. In detail the transfer process works as follows:

- An instance of an agent from the serialized data state is produced by feeding it into the `AgentInputStream`.
- The `AgentClassLoader` tries to load the appropriate classes from the *local filesystem*, first.
- If this is not possible, it tries to load them from the *agent's codebase*. Therefore the `AgentClassLoader` is provided with the address (IOR) of the agent's codebase. The codebase of an agent is the agent's home AP and the codebase is accessed through the method `fetch_class` of the home AP's `MAFAgentSystem` by the `AgentClassLoader`. From there the classes are loaded from the codebase that is distributed over the places within the AP and can be obtained from the `place` components that represent the places. The `AgentSystem` looks up the `Place` from where the class is available in its list of classes and fetches the class from there. Then the receiving `AgentSystem` is notified by the `Place` about the agents new address and reports it to the sending `AgentSystem`. This informs the `AgentSender` via RMI about the new address and then the messages are sent to the new location as already described. Internal transfer is carried out in an similar manner but the `AgentSender` sends the agent to the desired Place without involving the `AgentSystem`.
- If the classes are not available through these techniques, an *exception* to indicate the failed transfer is thrown back to the `AgentSender` by the involved Components (`AgentSystem`, `Place` etc.). Afterwards the agent is re-registered with the AMS and its execution is resumed at its current place.

The process has to be refined if the agent system that provides an agent's codebase is not reachable. An additional problem has to be solved if code is locally available but out of date or due to a wrong version. As this could lead to unpredictable behaviour of mobile agents, this issue has to be solved when determining the availability of code in the steps described above.

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

5 Related Work

Another MASIF and FIPA compliant AP called `Grasshopper` [1] has already been released in 2000 by GMD Fokus. For this platform two open source addons – one for MASIF and one for FIPA compliance – are available. But the approach to achieve this compliance has been started from the view of MASIF as `Grasshopper` was only MASIF compliant first and the FIPA addon has been released later. This addon complies to the FIPA 97 standards. At the time we evaluated the platform, the MASIF interfaces were not fully implemented in the MASIF Addon. For example it is not possible to use the MAFFinder for searching agents by their properties. Moreover, a `Grasshopper` agent system can only comprise one host. All places of an agent system are co-located on a single machine. So load balancing within one agent system is not possible. The approach described here is to focus on both standards to benefit from opportunities of both standards.

6 Conclusion

Interoperability on the agent system level is desirable for administrators responsible for many APs of different vendors. This paper shows how the standards of FIPA and OMG for multi-agent systems can be combined conceptionally and practically based on an *open source* AP. In this way interoperability on the agent communication level as well as on the agent system level are made available.

Although very important, *security* in multi-agent systems and for mobile agents remains an open issue for our future work because still FIPA has produced no specification that can be regarded as a standard and the suggestions of the OMG in its MASIF Standard can only be applied to some environments where a secure implementation of an ORB is available. Therefore these mechanisms have to be generalized so that they can be used in every environment. In some environments it may be required to have a `HTTP` interface on the agent system level as it is the case for the FIPA MTS that can use HTTP as a MTP. For the implementation security mechanisms are also an open issue and have to be developed. Mechansims to ensure timeliness of mobile agents' code and to obtain mobile agents' code when the codebase is not reachable have to be implemented.

[1] http://www.grasshopper.de

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

References

[1] N. R. Jennings (Southampton 2000) "On Agent-Based Software Engineering" Artificial Intelligence, 117 (2) 277-296.

[2] Foundation for Intelligent Physical Agents, *FIPA Agent Management Specification, Document No. 00023* (Geneva: FIPA, 2002).

[3] Foundation for Intelligent Physical Agents, *FIPA Agent Message Transport Service Specification, Document No. 00067* (Geneva: FIPA, 2002).

[4] Foundation for Intelligent Physical Agents, *FIPA Communicative Act Library Specification, Document No. 00037* (Geneva: FIPA, 2002).

[5] J.R. Searle, *Speech Acts* (Cambridge: University Press, 1969).

[6] Foundation for Intelligent Physical Agents, *FIPA ACL Message Structure Specification, Document No. 00061* (Geneva: FIPA, 2002).

[7] Foundation for Intelligent Physical Agents, *FIPA SL Content Language Specification, Document No. 00008* (Geneva: FIPA, 2002).

[8] Foundation for Intelligent Physical Agents, *FIPA Request Interaction Protocol Specification, Document No. 00026* (Geneva: FIPA, 2002).

[9] Foundation for Intelligent Physical Agents, *FIPA ACL Message Representation in XML Specification, Document No. 00071* (Geneva: FIPA, 2002).

[10] Foundation for Intelligent Physical Agents, *FIPA Agent Message Transport Envelope Representation in XML Specification, Document No. 00085* (Geneva: FIPA, 2002).

[11] Foundation for Intelligent Physical Agents, *FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification, Document No. 00088* (Geneva: FIPA, 2002).

[12] Foundation for Intelligent Physical Agents, *FIPA Agent Message Transport Protocol for HTTP Specification, Document No. 00084* (Geneva: FIPA, 2002).

[13] Foundation for Intelligent Physical Agents, *FIPA Agent Message Transport Protocol for IIOP Specification, Document No. 00075* (Geneva: FIPA, 2002).

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607

[14] Foundation for Intelligent Physical Agents, *FIPA Agent Message Transport Protocol for WAP Specification, Document No. 00076* (Geneva: FIPA, 2002).

[15] emorphia, *FIPA-OS Developers Guide* (Harlow: emorphia, 2001).

[16] Object Management Group, *Mobile Agent Facility Specification* (Needham: OMG, 2000).

[17] Foundation for Intelligent Physical Agents, *FIPA Agent Management Support for Mobility Specification, Document No. 00087* (Geneva: FIPA, 2002).

[18] Foundation for Intelligent Physical Agents, *FIPA Policies and Domains Specification, Document No. 00089* (Geneva: FIPA, 2001).

This is a draft of the paper published in: M.H. Hamza (Editor), *Proceedings of the 7th International Conference on Software Engineering and Applications*, 2003
http://www.actapress.com/PaperInfo.aspx?PaperID=14607