Electronic Contract Negotiation as an Application Niche for Mobile Agents[1]

Frank Griffel, M. Tuan Tu, Malte Münke,
Michael Merz, Winfried Lamersdorf
University of Hamburg
{griffel,tu,muenke,merz,lamersd}@informatik.uni-hamburg.de

Miguel Mira da Silva
University of Évora
mms@dmat.uevora.pt

Abstract

In this paper we propose electronic contract negotiation — a sub-part of the more general area of electronic commerce — as an example application for mobile agents. We start by presenting a set of requirements that an application should fulfill in order to take advantage of mobile agents and show that not all distributed applications meet these requirements. We then present contract negotiation, an example of an application that can potentially take full advantage of mobile agents. The paper also contains an introduction to a mobile agent system being developed at Hamburg University and show how it can be used to implement contract negotiation.

1. Introduction

The mobile agent (MA) paradigm has been gaining increasing attraction for the past 4 years. Its anthropomorphic idea is appealing to designers of distributed software systems: software programs that are able to migrate autonomously across the network during execution and to communicate with other agents and their hosts so that they can fulfill tasks on behalf of their principals.

Despite all this interest and consequent research efforts which even have produced some commercial products, *there is a constant lack of applications for these mobile agents*. One reason is that agents appear in a very competitive market: there are many other existing and proved mechanisms for cooperation and coordination in distributed systems, such as message passing, remote procedure call, code on demand [3], tuple spaces [22], or remote execution (e.g. stored procedures).

In fact, mobile agents have not gained any significant share of today's middleware market for developing distributed applications yet and before we really start the paper, it is interesting to analyze their main weaknesses and compare these with other (existing) technologies:

Mobile agent technology is not mature enough. There are several open issues that have not been addressed sufficiently yet. Examples include security, termination, persistence, or how to tackle network (remote) references. Although several MA systems that have been implemented by the research community and some commercial products are now available, e.g. Voyager [24] or Telescript [13], these issues still need more investigation. Furthermore, activities of standardization consortia such as the OMG have resulted in work on facilities for mobile agent systems [25]. So, instead of solving the problem ourselves in this paper — for some hints to our own approach see section 5 though —, we just assume that agent technology will eventually mature and converge to a standard model (as happened to RPC within the CORBA context).

Mobile agents require a large effort to be integrated into legacy applications. The execution environment for mobile agents may require additional setup effort in order to prepare the local computing environment to participate as a host for mobile agents. For example, an engine needs to be installed and local applications need to be "wrapped" such that they can be accessed by visiting agents and exchange data with them. However, an engine may be installed at run-time if it is required, by using Java as the programming environment (except for a bootstrapping kernel). When interfaces between mobile agents and local applications finally become a standard, this adaptation effort will probably be much reduced.

Application domains where mobile agents will succeed have not been identified. The existing communication and cooperation techniques are already suitable for most distributed application areas. In this context, the MA

[1] Appeared in: Proc. EDOC - International IEEE Workshop on Enterprise Distributed Object Computing, Surfer's Paradise, Australia, Oct. 1997

technology has to prove higher efficiency or lower costs, something that somehow justifies the setup costs of deploying an MA environment.

The first issue will probably be solved by itself as research goes on and commercial products are refined. Regarding the second issue and assuming Java as the programming environment, it will also become more and more irrelevant as Java and its associated libraries (RMI, JDBC, etc.) become ubiquitous. In this paper we will then concentrate on the third issue and, in particular, *identify a setting of technological and economic parameters under which an 'economical niche' exists for mobile agents*.

The rest of this paper is organized as follows. The next section presents an overview of mobile agents to emphasize issues that are important for this paper. Section 3 proposes a number of requirements for applications that can benefit from mobile agents and explains why not all distributed applications are suitable for mobile agents. Electronic contract negotiation, the example application, is then described in section 4, together with a description how agents can be used in this context. In section 5 we give a brief overview of the OSM agent system, an architecture that is being built at Hamburg University to support mobile agents and applications based on them. Finally, in section 6 we show features of these agents being used as "active contracts" to support a contracting process. The paper concludes with an outlook on our future work in section 7.

2. An overview of mobile agents

We define a *mobile agent* as an encapsulation of *code*, *data*, and *execution context* that is able to *migrate autonomously* and *purposefully* within computer networks *during execution*.

An agent is able to react sophisticatedly on external events. It may be persistent in the sense that it can suspend execution and keep local data in stable storage. After resuming activity, an agent's execution is continued — but not necessarily at the same location.

Mobile agents are alleged to provide suitable techniques for the implementation of electronic market systems [5,9]. These systems allow both demanders and suppliers of services and goods to exchange them freely based on electronic contract settlement and execution. In this paper we will use this application domain to determine the minimum set of facilities required from the mobile agent system.

2.1. Execution state

In the following, the term *persistent execution state* — as a postulated basis for migration technology — means that the agent should comprise a control flow definition at a coarse level of granularity. This allows not only persistent programming languages like Napier88 [26] and PJava [1] to be capable of representing mobile agents but also conventional systems like plain Java which may be able to provide only object-level persistency.

The main difference between these approaches is a matter of granularity at that an execution may be interrupted, migrated, revived and continued. An object-level persistency allows for choosing individual methods of an object as a control hook (continuation points after the object has been revived). This kind of "persistency" can be achieved right out-of-the-box by usual serialization techniques (like those offered by Java [27]) avoiding additional tools and proprietary environments.

On the other hand, truely persistent languages allow for more fine-grained execution hooks, even within individual methods. Having individual object persistence at a micro-level may not be enough to save the whole execution extent of an object system.

Here, we argue that the limitation to an object level does not really impose any constraints on the agent functionality since a reasonable activity usually comprises at least one method invocation — a high-level operation. Also, we like to avoid complex infrastructures like complete object stores — having in mind the potential "Internet consumer" sitting in front of her favorite browser doing home shopping.

On the other hand, recognizing the need for saving relationships between objects — especially in the case where a mobile agent's internal structure and/or the data it carries have to be quite complex — we will propose a lightweight solution to this problem by introducing the OSM profile in section 5.1.

For now, the conclusion from this seems to be a broad possible spectrum for mobile agent implementations spanning the two extremes:

- Agents as first-class values in persistent programming languages [17]
- Agents as data objects in traditional languages (including Java) [5, 24, 37].

The latter approach has a certain similarity to 'programming by composition'. For example, to factor out control flow definitions, workflow-management systems use petri nets or finite automata, etc. (at the *macro level*). The so-called "agent" is actually just a data structure that is interpreted by each (local) engine in order to invoke a coarse-grained function at the respective network site. This approach does not require any sophisticated language support, however, the main disadvantage lies in the necessity to factor-out any program logic to the stationary server — even simple arithmetic operations.

A well-balanced compromise of the previous two extremes would blend the advantages of mobile code with those of persistent data structures that determine control flow. In this case, local operations can still be performed efficiently by the agent code, but migrations will only be possible from distinct migration hooks that define the next entry point and indicate to the executing engine that a transfer of the agent is requested.

2.2. Examples of mobile agents

Having such a large design space for MA systems, it is interesting to take a brief look at some existing architectures. This is by no means a full review but should rather give the reader some hints for further reading.

One of the first Java based MA systems widely used by many researchers now is the MOLE architecture [27]. One of its main disadvantages is the inability to transfer executable code within exchanged messages but using special ”code servers“ instead. Besides adding another infrastructural component, this requires additional communication.

Of course Telescript, the very early commercial approach to MA systems, still influences emerging architectures. One interesting, often missed point is the fact that it actually does not move *code* but only execution *state*, assuming the code which this state belongs to is right in place.

Another interesting architecture is the Ara platform [29] offering a fine-grained view on execution flows. The developers promise the ability to migrate at any point within the execution, actually resulting in migrating processes. Since a further goal is the use of different programming languages with this system, the resulting object extent that have to be moved may get quite large raising communication costs.

The IBM Aglets [5] system also is a quite famous MA system now. There are some severe restrictions of this Java based architecture regarding the feasibility of message passing and collaboration. Aglets only allow sending string encoded messages to stationary agents at formerly known URLs. Also, one cannot move regular Java objects with this system.

A relative new approach can be found with Objectspace’s Voyager that promises to overcome some restrictions of other known systems. Besides addressing some problems of traditional distributed object programming (e.g. garbage collection, mobility, replication), this system aims at supporting mobile agents using Java again. Especially, the messaging system is very flexible, offering oneway, synchronous and future semantics. The coupling between regular objects and agents and their communication possibilities is very tight, but the persistency problem is not really covered. Voyager does not comprise a complete agent infrastruture but rather offers the building blocks to develop a such one.

Pjava [1] naturally has drawn attention to it as a potential basis for building MA systems. But it has to prove its suitability yet, especially, one has to take a close look at its persistency model. For now, no agent system based on it seems to exist.

Finally, there is our own mobile agent system, described in some detail in section 5.

2.3. Application domains for mobile agents

Many suggestions of potential application domains for MAs emerged from the research community as well as from industry. Our point is that although most (or all) of them are perfectly realizable by exploiting today’s technology, doing so is not very meaningful in every case or could be done with traditional techniques.

Suggested areas for MA usage comprise network and configuration management as well as information retrieval or quality of service analysis and assurance. Further fields are controlling manufacturing processes, supporting multimedia, or overcoming heterogeneity.

In the next section, we try to identify some measurable aspects that may allow for better estimation of mobile agents’ suitability in such application domains.

3. When to use mobile agents

In this section, we will identify several factors that influence the applicability of certain technologies on building distributed applications. Of special interest is the use of mobile agent technology in an electronic commerce context [10,18].

3.1. Requirements

This list of factors is neither exhaustive nor does it provide an exact quantitative means to formalize a utility measure for mobile agents. The resulting multi-dimensional space is just used to seek an area where MA appear as the most suitable technology.

Communication and bandwidth costs. We have always assumed that our agents will “roam the Internet” but Internet access is possible through several channels today. Some users will use a modem dial-in at zero telephone costs and at a flat rate of ca. 20,-- USD. This mode allows communication at a fraction of a cent per minute. On the other hand, communication costs increase when access takes place through cellular phone networks such as the GSM protocol for mobile phones in Europe. Here, tariffs vary between 0.05 USD and 1 USD per minute, depending on the different price policies. At the extreme position, a satellite-based system such as Iridium [30] may be considered, where one can expect per-minute costs of up to 3,-- USD.

Proportionally inverse to the per-minute costs is usually the available bandwidth for data communication. For example, an ISDN line allows for bi-directional communication at 64 kbit/s. GSM supports 9.6 kbit/s and in the case of Iridium a throughput of 2.4 kbit/s is most likely.

Therefore, the cost per kilobyte/s seems to be a suitable cost measure for this consideration. In table 1 we show the broad range of communication costs depending on the underlying technology. We assumed an average agent size of 100 Kb, including code, data, execution state, and serialization overhead).

Technology	Bandwidth (kb/s)	Cost per 100Kb ($)
ISDN	64	0.01 - 0.05 (15 sec)
GSM	9.6	0.08 - 1.6 (1.6min)
Satellite	2.4	21 (7 min)

Tab. 1: Range of communication costs

Although communication tariffs vary between countries and providers, this comparison provides a survey of the possible cost spectrum. It further considers the net transfer time of the agent, not idle time without data transmission.

Agent size. The size of the migrating agent strongly influences communication costs. Therefore, its size should be reduced as far as possible but without compromising too much the local processing time, storage space and other costs. We can distinguish between two factors that influence the agent size: the complexity of the carried agent application code and the underlying MA technology.

If the agent requires a huge amount of program code, even the "best" MA technology won't be able to effect a significant reduction in code size compared with the others. (The same is true for normal applications.) However, taken an application with a small share of code, MA technologies significantly vary in the size of externalized data objects. If the MA platform uses a commercially available object database system such as ObjectStore [31] then the overhead will be tremendous since the physical layout of object stores is optimized for rapid disk access and an efficient mapping to the physical disk. This may lead to a store size of several hundred kilobytes although no application data has been stored yet. For example, an "empty" Napier88 store has been 15 MB in size but in Pjava it is now 500 KB. As an additional factor, pre-allocated index tables can further increase the size.

Object serialization [27] marks the opposite end of the spectrum: an object structure that has been externalized to a standard representation usually reduces the overhead to 10-50% even when accompanied by a lightweight meta-information container like the profile presented in section 5.1. It is important to note that one cannot dispense with this additional information if consistent inter-object relations have to be guaranteed after remote revival. Therefore, there are no further savings regarding the overhead are possible if we want a full-featured MA system. For example, the minimum agent size that could be achieved with OSM agents at Hamburg University is 20 KB taking notice of the movement of all Java application classes — no code (despite the agents' engine) is therefore presumed at the remote target. A compression may further reduce this to 8 KB. Therefore, *the range in size of an agent typically extends from 8 KB to several 100 KB*.

Concerning agent technology, we also have many choices. For example, we can transmit plain byte-code or compressed byte-code as the Java JAR technology does. We can transmit machine code if the architectures are the same on the source and target machines [7]. We can even transmit part of the agent or none at all (just a reference possibly as small as a byte) if its code was already copied in a previous migration, or we can use "migration by substitution" [16,17] if the agent uses standard libraries or previously copied pieces of code and other constant data.

There are also many intermediate schemes between transmitting the entire transitive closure (as we propose) and executing the agent on the source machine as Kato [32] proposes, either specified by the application programmer or done automatically by the system [17]. However, it should be clear that all these compromises increase the computational complexity, CPU and space costs, or both.

Number of hosts involved. In this paper we roughly distinguish between N = 2 and N >= 2, where N is the number of machines involved. In the case of exactly two parties the agent doesn't need to maintain a node list since it just migrates to the "other host". In the second case the agent system needs a certain logic that determines the next target for migration. This address may be part of the agent code or obtained during execution.

The case in which N = 2 can therefore be subsumed under the category of remote execution. Since in this case the only (one) continuation hook is known formerly, there is no need to distinguish a choice of entry points. Therefore, the N = 2 case mainly resembles the situation conventionally found in remote method call scenarios: The one transition in execution triggered by moving to the other host could be just a remote call to that exact host.

Mobile agents are thus much more useful in situations where communications between N > 2 partners are required to be coupled only in an indirect way (stipulated by ad-hoc decisions, set-up costs or organizational policies, for example) not allowing for calling each other directly.

Value of the commercial transaction. Generally speaking, a higher transaction volume allows for higher transaction costs: in the case of 'micro-transactions' we assume a transaction volume of up to 5 USD. Under this assumption, transaction costs (i.e., costs that arise from the execution of a transaction, not from the product price itself) are tolerable when they remain significantly less than 10% of the transaction volume. Therefore, synchronous communications that require several minutes to complete a commercial transaction turn out to be commercially unfeasible. MAs, however, help to reduce these costs significantly since communication links are established only for the asynchronous agent transfer.
But even if we assume four transfers of agents that weigh 100 KB between business partners, communication costs of up to 1,6 USD drive the transaction value up to at least 16 USD.
This implies that an MA infrastructure may only be used either in the context of inexpensive broadband access or in mobile communication environments for high-volume transactions. For the first case, we stated that low communication costs also allow for synchronous or non-agent-based communication with out significantly higher costs. However, in the second case of higher transaction volumes MAs appear beneficial due to their better bandwidth utilization.

Relaxation of ACID properties. Distributed transaction support can be used to enforce consistency in order to assure the ACID properties (Atomicity, Consistency, Isolation, Durability) for all peers involved [33]. However, this semantics is acquired at the costs of communication overhead and reduced efficiency: the underlying transaction protocol requires a tremendous number of message exchanges if each agent transfer is to be secured under a transaction. For example, the 2-phase-commit protocol requires an overhead of at least 2*N messages for each application message that is sent to N nodes.

But it does not have to be like that. The overhead may be reduced for the application field of asynchronously migrating agents [34], yet it appears as a realistic estimation that transactions will duplicate or triplicate the number of messages required.

As a conclusion, *applications that are recoverable without transaction support appear to fit better to the MA approach* — as to any other distribution technology — due to the reduction of communication costs. Despite this, considering splitable and joinable agents like those presented in section 5.4, at least some of the advantages of transactional behaviour may be utilized, e.g. the assumption that some action irrevocably took place (see also section 6).

Agent execution environment. In order to allow arbitrary partners to cooperate by using mobile agents, the setup cost of the required execution needs to be minimal and, if possible, none at all. Systems such as Telescript or other systems that depend on proprietary technology require particular knowledge and effort for setting up a local environment (an engine). Ideally, the agent environment should be loaded on demand and use a programming system that is supported by every computer or at least easy to acquire and install.

Today, Java seems to be the most available and appropriate language to match this requirement. So we propose that an agent system intending to be used in real application contexts should be built on top of standard Java and its standard libraries (such as object serialization, RMI and JDBC) or at least equally ubiquitiously available software.

Level of autonomy. The literature on mobile agents often introduces autonomy as one of the indispensable features of mobile agents. However, if one argues in the context of a requirement definition, it should be phrased slightly different: if the application context required control activities between principal and agent, other communication paradigms may suit better. Tasks that can be accomplished by an isolated agent and the logic of its code appear as the domain for mobile agent applications.

Two different levels of autonomy should be distinguished. Firstly, the level of *communication autonomy* which concerns inter-agent communication, migration, and application access. Secondly, *decision autonomy* concerning the communication results. The second case implies knowledge of agents concerning the universe of discourse. This autonomy, however, falls into the domain of intelligent agents and is not further investigated in this paper although there is some ongoing research at Hamburg University's distributed systems group concerning "intelligently" negotiating agents.

Level of repetition. A tight coupling between two business partners justifies higher setup costs for the communication software. If, however, sporadic cooperation prevails, these setup costs may prevent any cooperation from taking place.

Mobile agents allow to ship the required application code to each respective network node. This makes the mobile agent approach more suitable for sporadic cooperation between distributed software applications.

Summary. Taking these factors into consideration shows that mobile agents are suitable for applications under the following conditions:

- High costs of bandwidth
- Small agent size
- Communication between many partners
- Medium to high transaction volumes
- Non-transactional communication support
- Ubiquitous execution environments
- High level of autonomy
- Sporadic communication

Obviously, the problem is to find applications that fulfill as many of these requirements as possible! In the next section we attempt to fill that gap by proposing contract negotiation as one such application. But prior to that, let us explain better why past attempts of using mobile agents have failed.

3.2. Design issues

Several example application areas have been proposed for the utilization of the mobile agent paradigm [8,9,35,36,37]. However, they still need to be validated against the requirements listed above. Approaches that raise the communication overhead also raise costs at the same time, so that the benefit of mobile agents compared to traditional approaches diminishes. Respectively, each measure that reduces bandwidth utilization should be evaluated in the light of these considerations.

In the mobile agent research field, efforts are now being spent on the fields enumerated below.

Management issues. Controlling remote software applications is already difficult, but controlling agents raises many new issues which in turn require additional communication overhead. For example, tracing the current location usually leads to a message exchange between the monitoring node and the current or past agent host. If this communication is considered as indispensable, then the designer may simply replace agents with an RPC mechanism or stored procedures. The benefit of mobile agents is reduced even further if active control is required for agent management.

External termination of agents. Sometimes an agent has to be stopped and terminated by a managing application that resides on a different host. This requirement first implies information about the agent's current location, secondly, it assumes that the agent's autonomy can be overridden by the principal's control. However, if the agent's task is characterized as too complex to be tackled by the agent software itself (e.g. if a certain level of autonomy is not given) — the mobile agent approach shouldn't be chosen, since one of the strengths of the MA approach is the ability to act on behalf of a principal which should include a meaningful termination behavior.

Security issues. Security has several flavors in the MA field. For example, agents can be authenticated when arriving at a place, the local engine may prove its benevolence through certificates, data may need to be encrypted on both sides, and inter-agent communication could be additionally secured against repudiation.

However, research efforts are still required in order to maintain privacy, authenticity, and non-repudiation. As an example, it is currently very difficult to avoid that an agent's privacy be assured against a hostile execution environment.

The more security support is enhanced, the more communication overhead has to be considered. Therefore, setting up an organizational or social environment for mutual trust seems to be more adequate than driving "security overhead" to the limits [38]. However, this should be understood as another niche definition: if the application requires a certain security level, mobile agents should not be used!

Payment methods. Finally, electronic commerce applications may require payment activities between agents. The need for payments indicates that agent communication takes place between organizations and possibly between business partners from different domains of trust. Taking that into consideration, if cash is used for sporadic commercial transactions the problem gets much worse if electronic coins can be accessed by hostile engines or agents.

On the other hand, if authorization is required from the paying agent, the communication overhead may again counteract the benefits of using mobile agents. Similar to the security question, designers of a mobile agent system should address this problem at the organizational level and not by technical issues alone.

Summary. As a conclusion, we could say that the design of a mobile agent system requires a very sensible and integrative consideration of organizational and technical solutions. Some researchers recommend means of social control [38] that help to punish the villains and to approve fair players among the agent society. In fact, a well-balanced combination of both aspects — technical and social — seems to be most promising.

3.3. A niche?

After evaluating the design space and illustrating some of the potential problems of „over-designing“ mobile agent systems, now the question occurs: What could be the niche for the mobile agent approach?

Generally, the MA paradigm is appropriate for all applications that coordinate sporadically established mobile user groups through asynchronous communication. For example, several games only need to pass control between human players and do not need to transfer extensive program logic and application data. Each player pays for the individual transfer of control by establishing a communication link for the transfer of an agent.

Another application field is the negotiation of contracts among business transaction participants: Here, a consortium is established that consists of individual participants who intend to agree on an electronic contract. This application scenario will be illustrated in the next section.

4. Electronic contract negotiation

Electronic contract negotiation falls in the much more general area of electronic commerce, which is itself an on-line version of business transactions either between companies or between a company and the general public.

Example of a contract. A South American company delivers coffee to a German importer. A French exporter delivers TV sets to the coffee supplier and the German company pays to the French one a certain amount of money. Such barter activities occur in international trade. The co-operation of multiple participants entails a chain of steps to settle a contract between them.

4.1. Business transactions

The subject of a business transaction is the exchange of goods or services between the parties involved. This could be a purchase where money is exchanged for a good. Generally, transactions may involve the exchange of money, services, or goods in any combination. The contracting parties may also exchange these goods in arbitrary order such that complex processes occur.

In general, a business transaction consists of three main phases (see figure 1).

1. in the *information phase* partners examine other participants' profiles or product offers, after which they either decide to:
   - *give up* (maybe to move on to another seller) or
   - *negotiate* the transaction;
2. during the *negotiation phase* certain properties of a service/product description are adjusted, and the outcome of this phase may either be:
   - the *abortion* of negotiation (end of transaction) or
   - an *agreement*; and finally after signing the agreement;
3. the *execution phase* is entered.

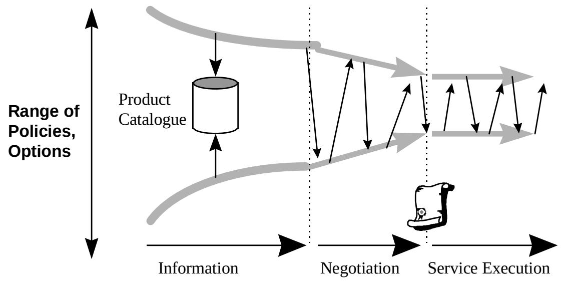

Fig 1.: The three typical phases in a business transaction.

4.2. An application for contract negotiation

An application for *electronic contract negotiation* is designed to support contracting parties in negotiating contracts.

Contracts as mobile objects. In this kind of application, a contract represents a data object that is accessed by the negotiating partners. During the negotiation phase, the contract data object circulates between the participants. Traditional concurrency control mechanisms coordinate accesses to the contract such that its consistency is preserved.

After receiving the contract, each party has the opportunity to change clauses or insert additional ones. A contract clause contains a number of standard informations:

- specifying the price of an offered service;
- describing the order of service exchanges between the partners; and
- defining the support services that are to be used by all transaction parties.

As an alternative, the contract could be stored at one host and be updated by any partners using distributed mechanisms such as RPC or Web technology. However, this centralized approach does not correspond to the whole idea of workflow that is fundamental to the contract negotiation, in which partners actually “own” the contract and explicitly send it to the other partners for review. A contract carrying agent can have a clear responsibilty and role for dealing with the contract while even avoiding explicit locking mechanisms.

Services to support negotiation. Support services [11] can be generally classified by the functions they supply. Some of these functions (payment and notary) have been involved in the scenario sketched in Figure 2.

Other services may be distinguished as complementary functions such as *quality assertion services*, which certify a distinct quality of service property to the client, or *protocol validation services*, which allow to restrict both client and server to a calling sequence (or *life-cycle*) originally specified by the server as a part of an augmented service description.

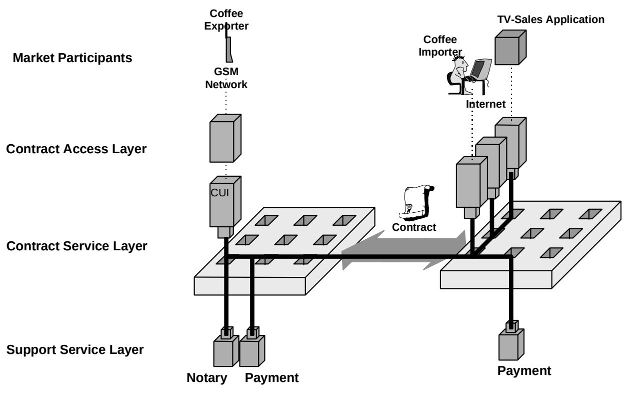

Fig 2.: Support services for contract negotiation

Many more support services may emerge that cannot be covered at design time by an electronic market system architecture. Therefore, a flexible naming schema and access method are required that allow for the registration of newly introduced *support service classes* at run-time [19].

The problem remains how to define the interface to each service so that an application that was not designed to use it can make use of it now that it is available.

4.3. An application niche for mobile agents

To illustrate the usefulness and applicability of MAs for contract negotiation, we take a more detailed look on how the scenario outlined at the beginning of this section may be supported by a mobile agent.

The South Amercian company wants to sell coffee, so they set up an agent giving him an initial contract template including offered quantities and indicate a barter deal by requesting to be refunded with color TV sets. This agent may then visit some online-marketplaces in search for coffee demanders, finding ultimately the German company.

The Germans indeed show their interest by extending the contract saying they like to have some coffee and are prepared to pay a lot money for it. For manipulating the contract's contents mechanisms offered by the agent are used ensuring consistent use and interpretation of negotiated issues. In this way, no concurrency problems accessing the contract can occur.

As a matter of the level of autonomy allowed — as part of the South American company's *policies* — the agent may deny the money offer and insist on delivering TVs by making a local decision or returning to South America where the contract's current state may result in the same decision (and the agent is again returned to Germany).

Since the Germans really like coffee, they think of ways to establish the contract. Therefore, they may tell the agent to visit some marketplace again in search for someone offering TVs for money. Now the agent finds the French company, shows her the current contract and asks if they like to participate. The French company's interest in the deal results in *splitting* the agent (see also section 5.3.) and sending the agents to Germany and South America respectively to signal its participation.

Both companies denote the expected buying and selling values (represented by money or TV sets) in their respective agents (remember the split!). Concurrency problems resulting from this shared editing of the contract may be avoided in two ways. First, the agent's protocol (see also section 6) could assure only *different* contract parts, i.e. those that are their ”own“, be edited by different participants. Second, a *unification* process [15] could be initiated when the agents join again — they do this by returning to the French company having a complete contract afterwards.

Alternative to participating the French company as an additional prime-contractor in the ongoing negotiation, the German company could also try to establish a *sub-contract* with it, hiding the existence of the coffee provider and eventually profiting more from the deal. However, such a strategy must be sophisticated enough to ensure both „halves of the deal“ and therefore has in principle a lower probability of success.

The last step of the whole business transaction is the settlement of the agreed contract. This could be supported by using the agent (still carrying the contract) as a controlling entity to a workflow system assuring the correct order of deliveries and refunding. There is a strong relationship between negotiation protocols, negotiation strategies and workflow descriptions that has to be explored further by ongoing research.

Despite a lot of missing details the above scenario should illustrate the principal ideas and some of the resulting requirements to a supporting architecture allowing for the application niche of electronic contract negotiation.

This niche exists since a (medium sized) company like the South American in the above example may only have access to low bandwidth communication lines. Thus, they cannot afford a full-time presence in the Internet. Also, the French company may offer such TVs only once (or sporadically) since it was a clearance sale. Therefore, it is not profitable to keep up an established communication infrastructure needing it just once.

Mobile agents seem to fit ideally into scenarios with such preconditions — which are not too unrealistic noticing that thousands of (even the smallest) companies will like to participate in the emerging global electronic market.

5. OSM: An implementation of mobile agents

In order to be able to experiment with the ideas described above, we have implemented a prototype of a mobile agent system based on Java that has been devised in the scope the OSM project at Hamburg University [20].

5.1. Service profile

The mobile agent system is based on a mechanism called *service profile* that is both a *persistent* and *movable* data object where the data may consist of code and some meta-information. The original requirement to the service profile was *the ability to describe any service that can be offered in the electronic market.* Each service profile comprises:

- the server interface specification to allow for remote access;
- service representation to the user, that is, GUI elements (leading to *self-presenting profiles*); and
- a specification of the valid method invocation sequences allows for checking (semantically) correct usage.

Since the profile is transferred to and processed at the user's local computer, it must be possible to serialize all profile elements. This means that the service profile in its current state can also be made persistent. The session with the server can therefore be resumed later, probably on a different host.

It must be noted that a control description can also be embedded in the profile. The form of the description may be chosen from a wide range of concepts (from a petri net assuming an evaluator up to coordinating Java classes) in a flexible way. Thus the profile can be used not only as a passive service offer specification but also as an active component in a workflow or mobile agent system, as will be described in the rest of this section.

5.2. Embedding agents

Although we have stressed the importance to use as many ubiquitous building blocks as possible for a widely usable MA system, this requirement cannot be fulfilled entirely since there has to be at least a certain level of standardized conventions. Nonetheless, for the sake of flexibility and autonomy, such conventions should be minimized as much as possible. Especially, a programmer of an agent should be free using any (Java) classes he wants for constituting the agent. Otherwise, the agent's personalization may be restricted or it may suffer from some functional constraints.

The OSM profile outlined above allows an agent to carry some meta-information about the classes it uses with it. Therefore, it is sufficient to agree upon the name of just one main class used to create the first object in an agents ”life“. This implies the following execution sequence for (re-)building agents. First, a class with a known (standardized) name, say ”Agent“, is loaded and instantiated. The new instance receives a reference to the profile it was extracted from to be able to build up the complete agent using whatever classes it (but not the environment!) knows about.

5.3. Agent migration

The mechanism described above means that all information necessary for a migrated agent to restore its old state and to continue execution is available to it at the remote location. Thus, the following agent migration procedure is supported. During its exceution the agent writes all information neede at remote locations into its profile. Since the agent writes the data itself, it also knows how to retrieve that information afterwards.

Then, bootstapping a migrated agent looks as follows. On the indication of an incoming agent, a remote agent-engine creates an assembler object that actually receives the agent and sets it up passing it references to the engine and the profile. If the agent needs some information from the environment it uses these references to contact the engine assuming some functionality to access environmental information is offered there. Due to this indirection, a better level of security is achieved since the locally installed engine can be specified, tested and controlled by local authorities.

An important feature of the OSM agents is their ability to *split*. Splitting means that an agent is cloned several times. However, local variables of each instance can be overwritten after cloning. Thus, depending on the state information the agents further execution may be carried out differently.

Often, some or all of the created ”subagents“ need to be synchronized for a later join to exchange information, for example about collected contractual issues. Therefore, an agent is able to identify another agent that it is wating for by means of that agent’s unique identifier — the way the OSM agent system names agents internally for now. Since it may not be desirable that an agent gains access to the information about what other agent are present at a specific location, this synchronization is done indirectly again, i.e. by a so called synchronizer object being part of the engine.

5.4. Agent communication

After being synchronized (no matter if it is with formerly splitted subagents or a completely different agent) agents may communicate. Since it cannot be predicted and all the more shouldn’t be fixed what information can be exchanged by the agents, a very generic communication technique is desirable.

A first idea is to require a reference to the potential communication partner (by asking the engine) and to cast this reference to the actual agent type. This approach assumes that the communicating agents are informed about their respective implementation details. Especially, in the case of formerly spitted agents, this is not too unrealistic. However, the current Java serialization package [27] does not allow casting such references to the actual agent’s type even if the types are equal. Thus, the current implementation uses a more pragmatic, nonetheless generic, approach by utilizing a method ”communicate“ with a parameter of type ”Object“. It should be noted, that since the agent communication does not involve third parties (like the engine), it can be kept confidential and be considered a secure channel. Credentials that authorize and authenticate one agent to the other could be passed as part of the given ”Object“ parameter.

6. Contracts as OSM agents

The described OSM mobile agent system is the basis for modelling contracts and their negotiation. The most important feature of the agents used is their layered architecture. Built upon a basic communication layer they include a strategy manager module and a negotiation protocol plug-in. The former for now is a policy [15] controlled pool of known negotiation strategies. We are planning to add dynamic action selection following the model of Maes [39] later. The latter is basically a finite automaton representing a distinctive negotiation protocol [40].

The idea is to ”plug“ such a protocol module into an agent. During a negotiation process the agent has to follow the represented protocol by all his communication messages being filtered through the negotiation protocol plug-in. Therefore, the agent is bound to a specific protocol and cannot ”cheat“ other negotiation partners that presume a specific negotiation protocol (since they obey it themselves). At the same time the plug-in assures (at least) syntactically correct negotiation messages. Such protocol modules may be offered and certified by a service of the electronic market system. It is important that an agent can trust such a plug-in since it becomes part of the agent’s communication channel.

Each negotiation protocol needs a *syntax* which is able to express certain message semantics. We have choosen a subset of the widely adopted KQML [4] for now, but may move to a more ”pragmatic“ solution in the future. Since the contract is the main item being negotiated in our application scenarios its contents are tagged using the same syntactical elements, e.g. ACCEPT, REJECT or DENY areas that could be filled in by the negotiating participants.

The contract’s contents may be arbitrarily complex since the profile’s genericity allows for great expressive power. The contract itself is just stored as ordinary Java objects. To keep things simple and running participants may use policies [15] — basically first-order predicate logic expressions in a lean disjunctive normal form supplemented by some modal operators — to denote their offers and requests. But the generic view on a contract as a container allows for many other possible ”description“ techniques.

In addition to the control of the negotiation process by the protocol plug-ins, the agents (since being active code) may observe the contracts they carry and assure consistency on their contents. Since at any time exactly one and only one agent ”owns its“ contract, no centralized concurrency control (e.g. locking etc.) is necessary for accessing the contract.

The described possibility to split an agent even allows to imitate some advantages of transactional behaviour. Unless a contractor (visited by a subagent) has filled in some required items in the negotiated contract the carrying agent would not arrive at a joining place. Therefore, if the agent returns, one can be sure that contractor has committed itself to some obligation. Simply speaking, an agent wouldn’t be there if it hasn’t done its job.

Two interesting points remain to be noted here. First, by utilizing the profile concept an agent not only can carry a contract but also a user interface customized to the contract’s contents as a visual aid for human users — an idea anticipated by OSMs *Generic Client* [20]. Second, regarding world-wide inter-organizational negotiation, the mobile agents autonomy helps collaborating across time zones. Thus, a negotation process may take place 24 hours a day by an agent migrating to locations during their individual business hours.

5.4. Evaluation

The criteria identified in Section 3 are applied now to evaluate the contract negotiation domain as an application niche for mobile agents.

- High *costs of bandwidth* are given due to the possible cellular phone connection of some participants or due to their low-bandwidth modem connection.
- If the agent is able to extract those contract items that are required or the target participant, it can keep a reasonable *size*.
- *More than two partners* are involved which pass the contract among each other in arbitrary direction and driven by a negotiation protocol.
- The *transaction costs* (Tons of coffee and TVs) are high enough to allow scope for communication across cellular phone connections.
- The overall contract negotiation may not be considered as a single transaction since inconsistencies are limited to manually correctable paragraphs of the contract.
- Java Virtual Machines are available for all technological platforms involved: PCs and cellular phones. The *ubiquitous execution environment* is therefore given.
- The agent migrates due to a negotiation protocol that is ”plugged-in“. Although this dependency counteracts autonomy, the agent still acts *autonomously* regarding the participants of the contract negotiation.
- The communication takes place sporadically, since this individual deal has not been done before and is not likely to be done in the future. Therefore, ad-hoc communications through MAs prevail.

7. Conclusion and future work

This paper is based on the hypothesis that “mobile agents” is a programming paradigm that is not applicable for all communication fields except for a very limited area in the space of all distributed applications. In this contribution we propose a set of economic and technical requirements that can be used to check whether an application can take advantage of mobile agents or should instead be implemented using some other technology of practical importance (message passing, distributed databases, CORBA brokerage, Web http).

We have then described an application that meets these criteria described before: contract negotiation for electronic commerce.

Ongoing research in the field of distributed systems support for electronic commerce at Hamburg University — called OSM — has been illustrated and finally how we plan to use OSM to implement on-line contract negotiation.

For example, the current mobile agent implementation already supports agent migration between computers and invocation of local functions. One of their specialized application that is currently being implemented is to present a user interface for contract access. This will permit to coordinate a group of transaction participants to settle a set of parameters and policies for the following execution phase.

By using the Java language, any Internet user with a Web browser is able to get involved in the contract negotiation process.

Although we decided to deploy a *useful first* approach [12] to create an architecture that is applicable to business scenarios right now by concentrating on the more technical aspects of communication, there is also ongoing research at Hamburg University concerning conceptual work to enhance mobile agents’ autonomy by giving them more ”intelligence“ [41]. Especially, negotiation abilities will be improved by self-adaptive decision making strategies and controlling policies [15]. Also, some further research to gain a clear understanding of the conception of persistence and its role in and influence on mobile agent architectures seems to be necessary.

References

[1] M.P. Atkinson, M. Jordan, L. Daynès, S. Spence, Design Issues for Persistent Java: A Type-safe, Object-oriented, Orthogonally Persistent System, In M.P. Atkinson, D. Maier, V. Benzaken, editors: *Proc. of the Seventh International Workshop on Persistent Object Systems*, Cape May, New Jersey, USA, May 29-31, 1996, Morgan Kaufmann Publishers, 1996.

[2] M.P. Atkinson, L. Daynès, M. Jordan, T. Printezis, S. Spence, *An Orthogonally Persistent Java*, SIGMOD Record, December 1996.

[3] K. Arnold, J. Gosling, *The Java Programming Language*, Addison-Wesley, The Java Series, ISBN 0-201-63455-4, 1996.

[4] H. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, G. Wiederhold: *An overview of KQML: A knowledge query and manipulation language*. Technical Report, April 1992.

[5] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Paris, G. Tsudik*: Itinerant Agents for Mobile Computing*. IBM Research Report RC 20010.

[6] Jordan, M.: Early Experiences with Persistent Java. In: *Proc. of the First International Workshop on Persistence and Java*, University of Glasgow, 1996.

[7] F. Knabe, *Language Support for Mobile Agents*, Carnegie Mellon University, Pittsburgh, PA 15213, USA, December 1995.

[8] E. Kovács: Advanced Trading Service Through Mobile Agents. In: *Proc. Trends in Distributed Systems '96*, Aachen 1996, pp. 112-124.

[9] T. Magedanz, K. Rothermel, S. Kruse: Intelligent Agents: an Emerging Technology for Next Generation Telecommunications?. In: *Proc. IEEE INFOCOM*, San Francisco, USA, March 1996.

[10] M. Merz, K. Müller-Jones, W. Lamersdorf: Agents, services, and electronic markets — how do they integrate?. In: A. Schill, O. Spaniol, editors, *Proc. of the International Conference on Distributed Platforms ICDP '96*, February 1996.

[11] M. Merz, T. Tu, W. Lamersdorf: Dynamic Support Service Selection for Business Transactions in Electronic Service Markets. In: *Proc. of the Intl. Workhop on Trends in Distributed Systems*, Aachen 1996, Springer, Berlin, Heidelberg New York 1996, pp. 183-195.

[12] O. Etzoni: Moving up the information food chain: Deploying softbots on the world wide web. In*: Proc. of AAAI-96 (Abstract of invited talk)*, 1996.

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

[14] T. Finin, R. Fritzson, D. McKay, R. McEntire: *KQML as an Agent Communication Language.* In: Proc. of the Third Conference on Information and Knowledge Management (CIKM '94), ACM Press, November 1994.

[15] M.T. Tu, F. Griffel, M. Merz, W.Lamersdorf: Generic Policy Management for Open Service Markets, accepted for publication in the *Proc. of the DAIS'97 workshop, Cottbus, Germany*, 1997.

[16] M. Mira da Silva, Models of Higher-order, Type-safe, Distributed Computation over Autonomous Persistent Object Stores, PhD thesis, University of Glasgow, 1996.

[17] M. Mira da Silva, M.P. Atkinson, A. Black, *Semantics for Parameter Passing in a Type-complete Persistent RPC*, Proceedings of the 16th International Conference on Distributed Computing Systems (Hong-Kong, May, 1996), IEEE Computer Society Press, 1996.

[18] M. Merz: Elektronische Dienstemärkte - Modelle und Mechanismen zur Unterstützung von Handelstransaktionen in offen verteilten Systemen. PhD thesis., University of Hamburg, November 1996.

[19] M. Merz, T. Tu, W. Lamersdorf: Dynamic Support Service Selection for Business Transactions in Electronic Service Markets. In: O. Spaniol, C. Linnhoff-Popien, B. Meyer editors, *Proc. TREDS - Intl. Workshop on Trends in Distributed Systems*, October 1996.

[20] Home Page of the Open Service Model project at the University of Hamburg, *http://osm-www.informatik.uni-hamburg.de*, 1996/97.

[22] Javasoft, Inc.: *JavaSpace Specification*, Revision 0.3, www.javasoft.com, March 1997.

[23] W. Lamersdorf: Datenbanken in verteilten Sytemen — Konzepte, Lösungen, Standards, Vieweg, Germany 1995.

[24] ObjectSpace, Inc*.: Voyager Core Package Technical Overview*, www.objectspace.com, March 1997.

[25] D. Chang, S. Covaci: The OMG Mobile Agent Facility — A Submission, pp. 98-110, In [42].

[26] R. Morrison, A.L. Brown, R.C.H. Connor, Q.I. Cutts, A. Dearle, G.N.C. Kirby, D.S. Munro: The Napier88 reference manual release 2.0. Technical Report FIDE/94/104, ESPRIT Basic Research Action, Project Number 6309 — FIDE$_2$, 1994.

[27] Sun Microsystems, Inc.: *Object Serialization*, www.javasoft.com, 1996/97.

[28] M. Strasser, J. Baumann, F. Hohl: MOLE - A Java based mobile agent system, In J. Baumann, C. Tschudin, J. Vitek, editors: *Proc. of the Second ECOOP Workshop on Mobile Object Systems (Linz, Austria, July 8-9, 1996)*, dpunkt, Germany, 1996.

[29] H. Peine, T. Stolpmann: The Architecture of the Ara Platform for Mobile Agents, pp. 50-61, In [42].

[30] Motorola, Inc.: http://www.mot.com/General/Events/TELECOM/95/Press/PR951001_44372.html

[31] C. Lamb, G. Landis, J. Orenstein, D. Weinreb. ObjectStore. *Communications of the ACM*, 34(10):51-63, October 1991.

[32] K. Kato, K. Toumura, K. Matsubara, S. Aikawa, J. Yoshida, K. Kono, K. Taura, T. Sekiguchi: Protected and secure mobile object computing in Planet, In: *Proc. of Second ECOOP Workshop on Mobile Object Systems*, July 1996.

[33] J. Gray, A. Reuter: Transaction Processing - Concepts and Techniques, Morgan Kaufmann Publishers, 1993.

[34] F. Morais de Assis Silva, S. Krause: A Distributed Transaction Model based on Mobile Agents, pp.198-209, [42].

[36] M. Baldi, S. Gai, G. Picco: Exploiting Code Mobility in Decentralized and Flexible Network Management, pp. 13-26, [42].

[35] L.A.G. Oliveira, P.C. Oliveira, E. Cardozo: An Agent-Based Approach for Quality of Service Negotiation and Management in Distributed Multimedia Systems, pp. 1-12, [42].

[37] J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel: Coummunication Concepts for Mobile Agent Systems, pp. 123-135, In [42].

[38] L. Rasmusson, A. Rasmusson, S. Jansson*: Reactive Security and Social Control*. http://www.sics.se/~ara/ index.shtml, September 1996.

[39] P. Maes: The dynamics of action selection. In: *Proc. of IJCAI-89*, pp. 991-997, Detroit, Michigan, August, 1989.

[40] T.W. Sandholm, V.R. Lesser: Issues in automated negotiation and electronic commerce: Extending the contract net framework. In: *Proc. of the First International Conference on Multiagent Systems (ICMAS-95)*, San Fransisco, June 1995.

[41] C. Sierra, P. Faratin, N.R. Jennings: A Service-Oriented Negotiation Model between Autonomous Agents, In M. Boman, W. Van de Velde, editors: *Proc. Of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'97, Ronneby, Sweden, May 1997),* LNAI 1237, pp. 17-35, Springer 1997.

[42] K. Rothermel, R. Popescu-Zeletin, editors: *Proc. of the First International Workshop on Mobile Agents, Berlin, Germany, April 1997*, LNCS 1219, Springer 1997.