Crossing organizational boundaries with mobile agents in electronic service markets

M. Merz, W. Lamersdorf
Hamburg University
Department of Computer Science
Distributed Systems Group
[merz | lamersd] @ informatik.uni-hamburg.de

Abstract

In today's network environments, a mismatch can be identified between existing, well-integrated distributed applications on one side and an increasing demand for spontaneous service access in electronic service markets on the other. In many cases, not only such buyer-seller relationships but also the management of distributed business procedures reaches beyond organizational boundaries. This often implies for most of the existing - tightly-integrated - workflow management applications that cooperating partners have to give up their local autonomy. However, emerging *mobile agent* platforms claim to provide suitable technical support to bridge this gap. To illustrate this approach, the *COSM (Common Open Service Market)* infrastructure is presented, which allows business applications to cooperate on one side but also to preserve their local autonomy on the other. Therefore, the mobile agent approach has been chosen for a flexible market-oriented integration of commercial services on the basis of the COSM infrastructure. This article shows how the basic COSM infrastructure is extended towards mobile agent support. Finally, an example illustrates this extension in the application field of interorganizational workflow management.

Keywords

Mobile agents, electronic service markets, client/server communication, interorganizational workflow management, petri nets

1 Introduction

An electronic market (EM) is defined as a computer-supported medium that allows both demanders and suppliers of goods to negotiate contracts at any time and without any spatial or temporal constraints (Schmid, 1993). Accordingly, the underlying technical system platform (the electronic market system) has to satisfy specific requirements that are also crucial for the coordination of demand and supply in real market environments. These include the possibility to easily enter the market as a supplier with new (innovative) services, to change product features autonomously, to easily access services as a customer, and to add value by enriching, combining, or customizing existing services (Merz, 1994a).

Cooperation in open networks

Open communication networks serve distributed software applications as well as human users as a common environment for the mediation and utilization of remote services. Beyond the scope of communication aspects (like, e.g., message delivery or protocol standardization),

distributed applications also realize cooperation in the sense of activity coordination in order to achieve a common goal beyond a syntactically correct exchange of messages. Within the scope of this paper, the term ‘cooperation’ is understood as (and restricted to) the client/server principle, i.e. a model which assigns to one of the cooperating applications a ‘demander role’, and a ‘supplier role’ to the other. Additionally, cooperation in open networks usually relates in this context to one of the two following integration levels:

- At a higher level of integration, only software applications act as requesters, i.e., they play the demander role by calling remote procedures on the supplier side. The underlying communication mechanism of this cooperation - remote procedure call, message exchange, function shipping (ISO, 1993), or a global tuple space approach (Carriero, 1992) - is not relevant in this context. What matters is the fact that in each case both client and server have to cohere semantically in order to engage in a meaningful cooperation and to constitute a distributed application. It is important to illustrate the context in which the cooperation partners have to ‘cohere semantically’: The corresponding (client/server) software components do not necessarily have to be developed by a common team; they have, however, to rely on a common convention - such as pragmatic agreements among developers ‘across the desk’, mutual specifications, conformance to a standard, or programming according to a common software documentation. In any case, in this application domain both developed components adhere to something in common - consciously or not, they form a *domain of conformance* across organizations. In the remaining part of this article, this class of applications is called (*organizationally*) *closed applications* of *classified* services.
- On the other hand, the lower level of integration always involves the *human user* who allows to relax conformance requirements for a given kind of service. Here, cognition and interpretation of ‘fuzzy’ application descriptions makes dedicated development of client software components obsolete. The World-Wide-Web (WWW) may serve as an example for such infrastructure: Interactive servers are used correctly in most cases if the provided service semantics is *obvious* to its users. As illustrated later, the communication technique does not matter in this case either: It may, e.g., be exchanged as an HTML document (WWW) by an asynchronous message transfer, or by a remote procedure call. However, in contrast to classified services, as addressed above, the message format and semantics is *not* standardized here. An interpretation of protocol data units does *not* take place after data transfer, therefore data is frequently represented in a plain text format. For the rest of this article, this class of interactively accessed applications is called *(organizationally) open applications* of *unclassified services.*

Of course, there is a trade-off between closed and open applications: In the first case, communication can be handled more efficiently, type-safe, and semantically coherent; in the second case, a service can be provided within a much shorter *time-to-market*, client applications do not need service-specific modules (stubs), and service providers are able to develop individual (therefore more competitive) functionality. In the latter context, early, innovative movers are better off. There is no general reason, however, to favor one approach over the other - both have their specific merits: Classified services are suitable for low-level, well established application domains like FTP, NFS, or the transmission of video streams - interactive services, on the other hand, fit well to user-level applications like ticket reservations or ‘kiosk services’. In the context of electronic service markets, however, sporadic cooperation predominates. Infrastructures that require high setup and transaction costs will therefore be a disadvantage. The main reason for the rapid proliferation of the WWW lies exactly in its extremely reduced

setup and transaction costs (in the sense of effort that is spent in order to establish a web server and to access it).

The mobile agent approach

The mobile (or itinerant) agent approach is a paradigm that has gained increasing attention with respect to the last point. Some emerging mobile agent (MA) platforms (e.g., White (1994), Wayner (1995), Harrison et al. (1995), Lange (1996)) claim to provide suitable techniques for the implementation of electronic market systems that allow both demanders and suppliers of services and goods to exchange them freely based on electronic contract settlement and execution. „Agents“ are metaphors used within a wide range of computer science research fields. As far as this article is concerned, however, only mobility aspects and the support of commercial interaction through MAs are examined in detail.

The migration capability is only partly given by Java applets, which allow to access remote servers on behalf of the user (Sun Microsystems 1995). Java applets thus provide a suitable abstraction from the actual interface and behavior of a server. Compared with the Java approach, however, MAs *actively* migrate to a customer’s local computer and allow for user interaction through a standardized GUI API. The only software component that provides a well-known programming interface is thus a commonly available access tool for agents users. Such a platform will be illustrated as a part of the COSM infrastructure.

As further elaborated in (Merz et al. 1996a), a typical MA scenario is given in Section 3, where third-party vendors of mobile agents supply added-value services that facilitate customer access to remote application servers. To give a practical example, the MA approach is applied to a Workflow management (WFM) scenario, where *ad-hoc communication between business partners* is addressed in particular. What remains is to allow providers to offer also *individual*, *unique* application services and to extend existing distribution platforms accordingly by means of access support to this kind of applications. For designers of electronic market platforms, a general question therefore addresses the gap between requirements of market dynamics and the capabilities of technical communication systems. This paper argues that the MA approach provides means to bridge this gap if correctly applied.

This article first examines the suitability of *MA platforms* to match the specific EM requirements for flexible service utilization. A corresponding realization based on the COSM infrastructure (Merz 1994a) is described in Section 3. For the rest of the article, we assume this electronic market infrastructure as a common platform to perform commercial transactions. An extension that applies mobile agents to support interorganizational workflow management (IOWFM) is demonstrated in Section 4. The COSM MA infrastructure serves as a communication mechanism here, which bridges organizational boundaries. Therefore, specific problems of (IOWFM) and the necessary extensions to the MA infrastructure are discussed in that Section as well. The final outlook sketches some further development directions for COSM and other future MA platforms.

2 Mobile agents

"Mobile agents" as a computer science research field reaches from distributed AI (e.g., Shoham, 1993), distributed programming (e.g., Tsichritzis (1987), Harrison (1995)) to the field of computer communications (e.g., Tschudin, 1993). This paper focuses on client/server coop-

eration aspects of MAs. In this context, approaches from research as well as from commercial products relevant to open service markets are examined.

Definition:

In the following, 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*. The *agent system* provides an algorithmically complete programming language. Therefore, an agent is able to show complex reactions on external events. An agent 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.

The given definition restricts the possible variety of agent implementation approaches to a quite small set of possibilities: Each case requires a local abstract machine - the *engine* - in order to execute the agent program. Regarding the two levels of integration analyzed above, the following application modes appear suitable for mobile agents:

- The most obvious agent application mode seems to be the *individual agent development* by the user: Agents may be programmed by the customer, for example, to perform database queries. This may result in a local full-text query or, e.g., a price inquiry for a specific good at each engine involved. Agents may also be developed to notify the customer. Here, messages are sent or activities triggered after a distinct event has occurred at a remote host. This may happen, for example, if an application state changes at the agent's site. Implementing such information gathering agents at the customer side requires specific knowledge about server semantics - therefore client and server form a *closed application* as defined in Section 1.
- Another rationale for agent activity is the provision of *added-value* through third parties: Existing - maybe heterogeneous - resources, like booking services, are accessible through an agent with a customized user interface or API. In this case, a (commercial) third party provides the agent as a *facilitator service* in order to support access to more complex service interfaces. A *facilitator agent* can be transferred to the client's site and bound to a local user interface tool. This enables users to access the facilitator service without using application-specific client software. Here, MAs help to bridge the gap between third party requirements of individualization (agent program) and infrastructure standardization (user interface components). This usage corresponds to the *open application* context.

Figure 1: Agents in the contexts of classified and unclassified services.

Mobile agents in open service markets

It is argued that an electronic service market emerges most dynamically if an infrastructure is provided that fosters proliferation of open applications.

Under these circumstances, MAs are best applicable if the domain of conformance is restricted to the service providing organization. The only possible way to decouple client and server in such a scenario is to install remotely a server-created agent on the client's side. The respective process may be executed either by the server as a 'remote installation' or by the client by 'agent acquisition'. In either case, such installation is only possible with the client's benevolence. The agent encapsulates all information and code required to allow human users

to interact directly with the agent itself or indirectly the remote service. This principle will be called the *server agent principle* (Figure 1).
To support electronic market applications, mobile agents offer certain characteristics that make this approach more favorable than, e.g., closed applications: Agents may directly support the application-level protocol required to invoke specific server operations. Today's service distribution mechanisms are, in most cases, still based on delivery of client applications, based on RPC or message exchange, with several inherent disadvantages such as the lack of persistent application state, arbitrary user interface styles, the burden of handling a large range of software applications with individual installation procedures on client systems, etc. Individual and spontaneous client/server cooperation is only possible if this burden is minimized by, e.g., dedicated facilitator agents that allow for the server agent principle.
Powerful *user interfaces* are required as well for a meaningful interaction between human users and agents. Therefore, a general client tool is needed to support service agent administration and user-agent interaction. In the case of MagicCap™ (White, 1994), e.g., a city-metaphor is used to browse a directory of agent offers and to support user access to the agents or services registered. Here, each service is represented by an individual building; user/agent interactions can take place after a building has been 'opened', i.e., an agent has been acquired.
In the following Section, the *COSM infrastructure* is introduced as a complementary platform for server agents. Designing and realizing an adequate system support for the access to remote services in open environments is the main goal of the COSM approach. This leads to a system software platform for realistic electronic service markets, which dynamically evolve and provide a great variety of individual service offers and requests.

3 Open application support in electronic service markets

As argued above, closed applications have several disadvantages in the electronic service market field. Most importantly, they require dedicated client components, which have to be installed individually. Therefore, the burden of system maintenance is shifted to the client users. On the other hand, current *open* applications, such as interactive WWW services, neither allow to store service descriptions (HTML documents) in a well-structured form nor do they provide sophisticated call-interfaces that conform to existing middleware platforms like CORBA (Common Object Request Broker Architecture; Object Management Group, 1993). Combining the advantages of both approaches - decoupled client and server development on the one hand, call interfaces with well-structured and well-processable interface descriptions on the other - leads to the development of the COSM platform (Merz, 1994a).

3.1 Providing user access

The COSM system platform has been designed and built in order to combine and support the advantages of both closed and open applications. The result is a flexible service implementation and access mechanism based on the concept of self-contained, generic *service representations*. A COSM service representation (SR) contains a set of description components defining, e.g., the operational service interface, the user interface layout, the relationships between user interface and remote procedure invocations, GUI interaction rules, comprehensive full-text descriptions of service functionality, billing information on the charge of procedure invocations, a definition of legal invocation sequences, etc.
The COSM platform consists of the following components:

- A *Generic Client* (GC) component that supports users in service discovery, access and binding, and in the inspection of service descriptions at run-time.
- A common *Service Representation,* which contains several service descriptions.
- *Service providers,* which implement application functionality through COSM servers. They are accessible on-line in the network. (They supply their respective SRs to potential users - either directly or indirectly through catalogue services).
- The *Object Request Broker* (ORB) which is used as middleware platform for DII-based (dynamic invocation interface; OMG, 1993) parameter transfer, and server activation. The SR is used as input for a stub generator only at the server's site.
- SR *Catalogues,* which let providers and users store and inspect SRs.
- Service *Traders* which identify appropriate service suppliers by an automated search for the best possible services based on given service description criteria.

Figure 2: The generic client as a service access tool

Based on this agent/SR analogy, the COSM platform can be extended smoothly with MA capabilities by simply augmenting COSM SRs: A COSM SR could thus contain both code and control flow specifications. In this environment, the SR corresponds to the server agent principle in the following way: The SR is developed and provided by the remote server. The user selects an appropriate agent from a distribution service such as an online catalogue. Agent and service are implemented within the same domain of conformance - whilst the client remains generic in order to allow its user to access any COSM server. After the GC received an SR, the corresponding user interface can be generated automatically at the client site. The user may then inspect the information provided by the SR and familiarize with the service functionality described. Finally, a direct client binding to the server can be established (or released if this information does not describe the kind of service the user was looking for). The user interface representation of a remote services site is standardized at every GC; GUI layout and content, however, may vary from service to service. This interface enables users to execute remote operation calls automatically by just entering data into forms and pressing corresponding buttons.
Before focusing on this task in more detail, the following section first elaborates a bit more on COSM SRs. SR extensions for control flow specifications based on petri net representations are illustrated thereafter.

3.2 COSM service representation

In COSM, the crucial vehicle, which contains and transfers arbitrary and location independent service descriptions, is the *service representation*. At the most generic level, the SR is a container for data structures of arbitrary run-time types. After receiving an SR, the Generic Client interprets the embedded description components. The following components are used in COSM:

- A specification of operation descriptions, containing operation names and parameter descriptions. Parameter descriptions refer to data objects which contain actual values. These components resemble roughly IDL clauses known from DCE or CORBA.
- A specification of the *user interface* that the GC generates for human users. It contains specifications for dialog boxes, data editors and push buttons. If attributes of the GUI specifications are used as RPC parameters or results, the presentation may be changed dynamically by the remote application (e.g., the visibility of dialog boxes, or the activation state of pushbuttons).
- A specification of the service interface protocol, i.e., which operations are enabled to be invoked at a given state. Currently, this protocol description is based on a *finite state machine* model and comprises a set of application states and transitions between them, which, in turn, refer to operation descriptions. State changes are caused by user-level events and RPC results.
- Informal description components as, e.g., help texts which, directly support human users.
- 'Price tags' that inform users on operation invocation costs. This information ranges from simple monetary values to sophisticated pricing policies - it depends on COSM applications how to represent price information.
- Any local data values used by the GC in order to represent the application state.

Since the COSM Generic Client is not restricted to a distinct set of applications, specific service state information, such as window positions or counter variables, may also be captured by the service representation. Furthermore, it is also possible to store the SR persistently and to suspend interactions with the remote server temporarily and resume them later. This is based on the stored SR status information on a remote network site.
Compared with HTML documents, the SR approach therefore facilitates storage and query processing for service descriptions since they can be stored in dedicated repositories. Type objects are used as further SR components for the dynamic extension of the repository database schema. This allows service providers to extend SR description by individual data items. On the other hand, it allows users to query against the repository such as: "Return all servers that provide an operation 'Book'", or: "Return all parameters of the operation 'Book' in the SR 'CarRental'".

3.3 The COSM mobile agent infrastructure

SR-based interface specifications do not invoke remote procedures directly; they are rather used to guarantee type-safe remote service invocations that are initiated by the generic client on behalf of a human user. Taken together as a unit, user, generic client, and server can be considered as an *engine*, that interprets SRs and executes the operations described. Users trigger invocations, and the server implements the respective function. If, e.g., a server agent is to be developed, the following extension is required for the basic COSM architecture: Remote services need *engine capabilities*, i.e., abilities to transfer and receive SRs and to locally execute some basic operations of the agent language that are common to all agents. In such an environment, e.g., the operation *Go* <*hostname*> transfers the SR including the current evaluation state to the receiving host and the operation *End* terminates the agent, by letting the engine delete the SR. Every other operation is not carried out by the COSM engine itself but is forwarded to a COSM server.

The integration of agents with the COSM generic client user interface also requires a repository for agents that are available to the user. For that purpose, a directory as being used by Telescript (represented by a city metaphor) is provided by an agent catalogue in COSM. Agents that are ready to be used can be obtained from that catalogue.

3.4 Embedding control flow into COSM SRs

The base model of COSM did not provide means to incorporate control flow specifications into SRs. This motivates the corresponding SR extension by petri net representations as presented below.

In the context of COSM, the basic petri net model (Jensen, 1992) is semantically extended by the concepts of *split* and *join transitions*: A split transition separates a single SR into two or more independently executable SR net representations. In contrast to regular transitions, split transitions do not mark each outgoing place with a token; they rather generate a copy of the net representation after tokens have been withdrawn from all incoming places and occupy only one single outgoing place per net instance instead. In COSM terms, additional *sub-SRs* (resp. subagents) are thus created out of the *master instance*. These instances can be processed independently until a *join* transition is reached. The join transition is not enabled unless all required instances have arrived at the input place. All net instances, except for the master, are removed when the join transition fires. To carry out a join, the required SRs have to be collocated at the same engine.

Each transition definition is associated with one operation description within the SR. If the transition predicate evaluates to *TRUE*, firing the transition causes an invocation of the associated operation.

Pairs of split and join transitions may be *nested* in net definitions. Therefore, a dedicated naming schema is required to assure correct identification of the agent instances to be merged. These agent instances form a common *task* that is set-up by an initial agent. Tasks belong to a *task type* and tasks of various types may be executed by an engine at the same time. A task type is defined by individual agent implementers. The resulting naming schema appears as follows:

1. *Task type identifiers* are declared by agent implementers. However, to enforce uniqueness, a type identifier comprises a universally unique identifier (UUID).
2. At each task creation (i.e. when an initial agent is instantiated), a globally unique *task identifier* is appended to the type identifier.
3. Each split transition, in turn, appends a locally unique *instance identifier* to the task identifier. The join operation is therefore required to strip off this identifier and to test the remaining names on equality. Of course, additional instance identifiers may be appended at each split. This schema enables an engine to merge the least recently created agent instances first.

3.5 Move on!

In summary, the basic components of the COSM based *engine architecture* are

- the *service representation* as an encapsulation of the agent's local state and control flow,
- the *engine* as an agent evaluator: The engine executes operation invocations and passes agents on to other engines. Application-specific operation calls are invoked at remote serv-

ers, which conform to the agent's service description. Every engine performs operation invocations at the same server.

- *COSM servers* provide application services at remote sites. Their operations are invoked by engines. Servers are not specific to the MA approach, they can still be invoked the 'traditional way' from generic or specific clients. Therefore, MAs as a distributed application concept remains compatible with the existing RPC mechanism of COSM.

Agent operations are distinguished into *well-known commands* and *application-specific* ones: Well-known commands can be executed locally by each engine while the latter are delegated to the COSM server that is connected to that engine. Such a service invocation is executed as if done by a generic client: First, the current Petri Net state is determined by the engine. If the predicate of the currently enabled transition evaluates to *TRUE*, the transition is fired. By means of the SR operation definition, a named value list is created and transferred to the COSM server through a CORBA DII invocation. Each invocation advances the local state of an agent.

Agent splitting and joining is carried out by the engine. A split transition definition within the SR is associated with an address list of target engines. When firing, the engine creates subagent instances - one for each element of the address list - which are transmitted to the respective target sites.

Concurrent agents are merged into a single instance if all of them belong to the same task and have arrived at the same engine. These instances are stored locally at the engine's site unless all of them have arrived. In this case, all except for the master instance are simply deleted.

Concurrently migrating agents that belong to the same task change their local state individually due to different operation invocations at different engines.

3.6 An example

Figure 3: Migration scenario for mobile agents

The scenario illustrated in Figure 3 shows migrating agents, which obtain price information from a set of services: From the *agent catalogue*, an agent is loaded *as an SR* through the GC, a user interface is created and initial data is entered by the human user. From now on, the SR is treated as an agent and is transferred to a *yellow pages server* by executing an according *Go* command. After an address list of suitable servers has been obtained, the agent splits into three instances: The *master agent*, and one subagent per target address in order to perform the *GetPrice()* operation at each server. These servers are available as unchanged COSM server that can still be directly accessed through the generic client as shown in Figure 2. After each agent instance has gathered its respective price information, a *Join()* is carried out on a fifth host - the agent provider host that implements task-specific aggregation functions for the individual results. For the example task given here, the resulting prices are simply added. Finally, the remaining master agent is redirected to the agent catalogue. From there, the user (who may remain offline for most of the time, e.g., in a mobile computing environment) obtains the final result through the generic client again (Figure 3).

The goal of the COSM infrastructure is to enable an independent proliferation of application services, value-added services, and extensions to service representations. The example discussed above, illustrates these possibilities: The agent acts as a mediator between the human user and a set of underlying reservation services for a restaurant and a flower shop. For this

purpose, the original SR has been extended by control flow definitions. These two extensions may take place without any central configuration or standardization. To follow this principle one step further, the application domain of workflow management will be considered as suitable testbed for COSM agents. In this context, both the value chain of services and the SR will be further extended by workflow coordination services and role descriptions of workflow participants.

4 Interorganizational workflow management with mobile agents

Workflows are typically executed in distributed computing environments, for example, in huge organizations or even in multi-organizational settings. Therefore, appropriate workflow management systems must be able to deal with both distribution and heterogeneity. Due to organizational modifications of the company, workflow management systems have to be scaleable in order to satisfy changing requirements when new business processes or departments need to be created, external organizations to be integrated, or existing services outsourced.

Workflow definition and execution

Workflow management deals with the specification and execution of business processes. General *process definitions* include the *activities* to be performed, their control flow and data exchange. They also comprise organizational roles of persons and software components that are permitted to perform individual activities. Policies, which describe the organizational environment, complete a process definition (Workflow Management Coalition 1995). The workflow management task can be generally separated into the phases workflow *process definition*, workflow *application configuration*, and workflow *execution*. In most cases, WFM systems are used in closed organizations where application interfaces and semantics are well understood and can be configured in order to satisfy given demands for conformity. (Some representatives of these systems are Mobile (Schuster et al. 1996), Meteor (Miller et al. 1996), and Exotica (Mohan et al. 1995)). Therefore, contemporary WFM systems either support a programming-level application integration or a workflow description notation that requires for its interpretation the installation of an expensive workflow execution environment. However, in an interorganizational context, applications that reside at external organizations can not be smoothly integrated, since neither their interfaces nor their semantics may be changed. Moreover, separated organizations maintain their local design and communication autonomy and - from an economic point of view - the setup costs would be too high if a remote application needs to be adapted only for sporadic accesses. Only at the level of generic browsers - as known form the World Wide Web - an integration of remote workflow participants will be feasible at moderate costs. The usual problems of interorganizational WFM systems are thus:

1. *Lack of a common cooperation infrastructure*: Separate companies may lack a shared cooperation platform - in fact, data exchange across telephone lines is the common denominator in many cases. They may operate by means of heterogeneous operating systems, application software, and even workflow management systems. An integration of these components would require high set-up costs which reduce the benefit of interorganizational WFM.

2. *Lack of central management*: On the market, participants coordinate their activities through the price mechanism - not by objectives that are communicated through hierarchical channels of a closed organization. Companies acting as trade partners on the market might not intend to tighten their cooperation up to the degree that is given, for example, in the automotive industrial sector. Companies might rather intend to integrate isolated tasks from separated companies into their individual processes. One may think of a manufacturer who wishes to initiate a quality improvement process in a supplier's production. Here the costs of calling the responsible manager by phone diminish compared to the set-up costs in the case of an organizational integration. Therefore, the WFM software infrastructure must be able to cope with such sporadically occurring events.

3. *High coordination costs of WFM systems*: Activities that aim at integrating interfaces of heterogeneous WFM software products have risen in the past years (WFM Coalition 1996). However, the result is an interoperability at a very specific level that requires both partners to invest into expensive equipment. Therefore, the WFM software - as a closed application - only pays off in settings with highly repetitive activities. Efficiency would not decrease if the level of interoperation is lowered to a more generic collaboration infrastructure allowing to integrate not only WFM applications but also an access to any other on-line services. In this case, the communication infrastructure is already given, and the WFM integration costs decrease to the extension of such an existing infrastructure.

As a conclusion, a conflict between open application requirements and the capabilities of closed application is obvious in the field of IOWFM. In this setting of a the sporadic communication with autonomous trade partners, the bottom line of common knowledge might just be the business partner's local agent platform. The following application scenario shows an example for a process that spans across organizational boundaries and therefore involves external business partners in an interchangeable manner:

4.1 An application scenario

In a trading company, an order processing task is initiated by a customer request (Figure 4). As the first step of internal processing, the warehouse manager is inquired to check whether the item requested is on stock. If this is the case, an offer is made and returned immediately to the customer. If not, a list of potential suppliers is provided either by a clerk or by a software component. This process may involve several other data entry activities as well.
Each supplier receives, in turn, an offer request and returns the corresponding individual offer. Again, either a program or a person selects the most appropriate offer and requests a financing offer from one of the involved banks. This is carried out by transmitting concurrently a GETFINANCINGOFFER to each bank server. After a distinct time-out or if all banks have replied their conditions, one bank is selected and, as the final step, an offer is made to the customer.
This process comprises different kinds of tasks that are well-structured like the ONSTOCK function. They could be well performed by conventional client/server tools. Other tasks - for example the decision, which supplier to involve - may require human activity in order to be accomplished properly. The decision to perform some tasks automatically or manually may be delayed until the process instance has reached the respective state of execution.

Figure 4: An application process scenario

The data structure or *substrate* that represents a process instance may either reside locally at each involved application or it is a mobile agent, which carries this information around as a payload. The first case refers to the basic COSM approach, where an SR is obtained from a central application server in order to perform remote procedure calls. In the latter case, all information that is associated with the process instance is encapsulated as an isolated data object, i.e., the SR of a mobile agent. It is important to mention that the process definition (the petri net from Figure 4) is independent from the implementation of the workflow execution.

4.2 Extending the COSM agent platform for workflow applications

If, e.g., a bank clerk is prompted to enter a financing offer in the scenario all activities should be disabled that are not allowed to be carried out by him. Further, the actual COSM *application server*, which, e.g., provides an interface to access stock information, is separated from an additional *task server* which manages a service representation repository and coordinates invocations of the application server. Users who intend to perform any task have to be authenticated by task server through their *role identifier*. The task server then provides an individual to-do list to the attached user with all process instances in an execution state that permits to perform an activity. As an SR, the process instance is acquired by the generic client in order to let the user access the application server through remote procedure calls. After this interaction has taken place, the SR is transferred back to the task server and might appear on the task list of another user (Figure 5). For this purpose of service access coordination, the same type of petri nets as defined in the MA scenario can be used. However, at process execution time, these control flow descriptions are not used to allow an SR to migrate around as an agent unless a *Go* operation is fired.
In this case, it could be immediately transferred to a remote engine if a transition with the *Go* command is encountered. In the agent-based example of Figure 4 the financial offer calculation for the trading company could be done by a user at the bank's local engine. Instead of accessing the trading company's task server, the local engine of the bank could enter the agent (as an SR) into a local task server. After a financing offer has been made, the SR is transferred back to this task server and from there (as an agent) back to the trading company.

Figure 5: Coordinating SR-based server accesses through the task server

4.3 The trading company example, revisited

After the COSM infrastructure, its MA extension, and, finally, a further augmentation towards IOWFM have been introduced, it is appropriate to combine all approaches:

- Our implementation scenario will therefore use the *generic client* in order to allow flexible access to remote servers (the trading companies application server),
- it will use *engines* to execute SRs as agents,
- a *task server* is involved, where process instances are kept as task list entries, and
- SRs - will be treated as *mobile agents*, if their transfer to a remote host is indicated through the *Go* operation.

To provide a further refined example, Figure 6 shows the process definition for this example. Split and join transition are defined to control subagent creation and collection. Engines carry out this task and treat SRs as agents. For external business partners, the task server acts as an engine that collects agents after an offer was made and the respective agents migrated back to the local organization. *Within* this organization, the task server treats these agents as an SR again.

Figure 6: Control flow specification

The advantage of this SR interpretation - either in the 'traditional' way or as a mobile agent - provides an additional flexibility of the process definition: It is independent of the control flow description (as illustrated in Figure 6) how an SR might be interpreted at run-time. Tools for process definition and execution are therefore independent from one another as well. If the task server of the example in Figure 7 is slightly modified in order to introduce engine capabilities, external participants may switch from the task-list approach to the agent based one. The control flow definition as a part of the SR will remain constant in both cases.

Figure 7: Combining the centralized and the decentralized approach

In this example, the service representation is considered as a workflow process instance and as mobile agent at the same time. Its interpretation depends on the application context - but not on the net definition. Therefore, the only required infrastructure is a generic client and a combined engine/task server function at each organization. To provide application functionality, COSM servers are required as well.

Due to the separation of engines, task servers, and application servers the overall agent environment allows for scalability of the coordination services: both the task server and the engine can be replicated as long as the external interface towards generic clients and remote engines remains stable.

The current prototype of the COSM MA system platform was developed in C++ for an IBM RS/6000 workstations with AIX. A second implementation exists for OS/2 with the option to run engines on both platforms and to exchange agents across them. The necessary heterogeneity transparency is achieved by a platform-independent implementation of SRs.

The interpretation of the SR used in the demo environment and the set-up of the GUI takes up to 5 seconds. The size of the MA-enhanced SR is ca. 50 KB, however the size can be reduced to ca. 6 KB by compression such that it is easily transferable across the Internet. Thus, the overall workflow execution takes less than one minute (delays caused by the human user are not counted).

5 Summary and outlook

Mobile agents are a promising approach to implement a platform for electronic service markets. However, standardization of user interfaces, service attributes, etc. becomes a sensitive design decision in this context. Therefore, one rationale for the agent-oriented approach presented in this paper is to identify an organizational model that fosters electronic service market proliferation.

The approach based on the COSM architecture aims to add-on MA facilities to the basic CORBA DII based client/server infrastructure. This keeps MAs as an option that the application programmer may utilize or not. The service provider's design autonomy is thus preserved. Embedding petri nets as a means for control flow specifications and for task coordination encourages third-party vendors to provide added-value services such as agent catalogues or task servers. These reasons for the selection of the MA approach emphasize the extensibility of COSM as an open distribution platform. The COSM infrastructure allows to purposefully extends an existing middleware framework.
It was further shown how the approach of mobile agents can be applied to the field of workflow management. Whilst involving mobile agents in an *intra*organizational context might appear to be less efficient due to economies of scale, it appears to apply well, however, in the context of WFM applications that span organizational boundaries. It was further discussed on the basis of a representative application example, which design options exist for a possible implementation of such a workflow environment that is based on an MA platform.
The MA approach helps to break up domains of conformance for closed distributed applications. It supports to gather and access remote services in a type-safe but spontaneous way.

Outlook

The discussed MA infrastructure has been implemented based on several already existing COSM concepts and components - the generic client, the service representation, the catalogue servers, etc.. The integration of additional security mechanisms such as *non-repudiation services* or the support *electronic payment functions* is a current subject of research that is discussed in (Merz 1994b and Merz 1996b).
The MA-based WFM infrastructure of COSM mainly lacks a flexible, individual implementation of application code. Only existing application servers are accessible and coordinated by the mobile agent framework of COSM. Additionally, local agent code for user interface control, result computation or subagent synchronization has to be externalized to these application services as well. Another shortcoming of the current implementation is the entire transparency of the information contained in the service representation. This concerns both control flow description and data.

As a solution, Java's flexible binding and loading mechanism combined with the given distribution transparency allows a smooth transfer of code. However, a shortcoming of Java is the lack of a persistent encapsulation and transfer of execution state between the engines. Two possibilities exist to accomplish this: Either Java abstract machines need to be enhanced for a persistent execution management (Atkinson et al. 1996); or Java libraries need to be developed, which allow application programmers to define persistent objects as well as a suitable migration mechanism (Lange 1996). The first case requires modified Java interpreters and would therefore reduce the infrastructure's ubiquity. In the latter case, the execution context may not be persistent at a fine grained level of individual abstract machine statements but rather at the coarse-grained level of defined methods that are executed after an agent arrived at the target engine and resumed processing. To achieve this in the context of the COSM infrastructure, execution state has to be integrated into the SR together with the original agent code (i.e., Java applets). Here, the service representation appeals as a mobile data store for both data and code persistently.

6 References

Atkinson, M.P., Jordan, M.J.Daynes, L., Spence, S. (1996) *Design issues for persistent Java: A type-save, object-oriented, orthogonally persistent system*. In: Proceedings of the Seventh International Workshop on Persistent Object Systems, eds. M. P. Atkinson, D. Maier, U. Benzaken. Morgan Kaufman Publishers, pp. 33-47

Carriero, N., Gelernter, D. (1992) *Linda in Context*. CACM, 32(4): 444-58.

Harrison, C. Chess, D., and Kershenbaum, A. (1995) *Mobile Agents: Are they a good idea?*. IBM. Research Report #RC 19887

ISO/IEC JTC1/SC21 (1993) Remote Database Access (RDA) - Service and Protocol. OSI International Standard 9579-2.

Jensen, K. (1992) *Coloured Petri Nets*. Berlin Heidelberg New York

Lange, D.B., Chang, D.T. (1996) *IBM Aglets Workbench: Programming Mobile Agents in Java*. White Paper. IBM Corporation, September 9, 1996, also http://www.ibm.co.jp/trl/aglets

Merz, M. Müller, K., and Lamersdorf, W. (1994a) *Service Trading and Mediation in Distributed Computing Systems*. In: Proc. IEEE International Conference on Distributed Computing Systems (ICDCS), Los Alamitos: pp. 450-457

Merz, M. Müller, K., and Lamersdorf, W. (1994b) *Trusted Third-Party Services in COSM*. In: Electronic Markets, eds. R. Alt and S. Zbornik, 4(12), St. Gallen, Switzerland, September: pp. 7-8

Merz, M. Müller, K., and Lamersdorf, W. (1996a) *Agent, Services, and Electronic Markets - How do they integrate?*. In: Proc. ICDP International Conference on Distributed Platforms, eds. A. Schill and O. Spaniol.

Merz, M., Tu, T., and Lamersdorf, W. (1996b) *Dynamic Support Service Selection for Business Transactions in Electronic Service Markets*. In: *Proc. Intl. Workshop on Trends in Distributed Systems*, Aachener Beiträge zur Informatik, Vol. 17, pp. 183-195

Miller, J.A., Sheth, A.P., Kochut, K.J., and Wang, X. (1996) *CORBA-Based Run-Time Architectures for Workflow Management Systems*, Journal of Database Management, Special Issue on Multidatabases, **7**(1), 16-27.

Mohan, C., Alonso, G., Guenthoer, R., Kamath, M. (1996) *Exotica: A Research Perspective on Workflow Management Systems*, Data Engineering Bulletin (Special Issue on Infrastructure for Business Process Management), **18**(1), 19-26, March 1995.

Object Management Group (1993) *The Object Request Broker: Architecture and Specification (CORBA)*. Specification 1.2, Object Management Group, Framingham, MA, USA

Schmid, B. (1993) *Electronic Markets*. Electronic Markets, 3 (9/10), 3-4.

Shoham, Y. (1993) *Agent-oriented programming*. Artificial Intelligence, 60, 51-92.

Sun Microsystems (1995) *Java: Programming for the Internet, Java & HotJava*, http://www.javasoft.com, White Paper

Tschudin, C. F. (1993) *On the Structuring of Computer Communications*. PhD thesis, University of Geneve, 1993

Tsichritzis, D., Fiume, E., Gibbs, S. , Nierstrasz, O. (1987) *KNOs: Knowledge Acquisition, Dissemination, and Manipulation Objects*, ACM TOIS, **5**(1), 96-112.

Wayner, P. (1995) *Free Agents*. Byte, 20(3) 105-14.

White, J.E. (1994) *Telescript Technology: The Foundation for the Electronic Marketplace.* White Paper, General Magic, Inc.

Workflow Management Coalition (1996) *Reference Model and API specification.* http://www.aiai.ed.a.c.uk/WfMC

Schuster, H., Jablonski, S., Heinl, P., Bussler, C. (1996) *A General Framework for the Execution of Heterogenous Programs in Workflow Management Systems,* Proc. First IFCS Int. Conference on Cooperative Information Systems (CoopIS 96), Brussels, pp. 104-113