Interorganizational Workflow Management with Mobile Agents in COSM

M. Merz, B. Liberman, K. Müller-Jones, W. Lamersdorf

Hamburg University - Computer Science

E-Mail [merz | 1liberma | kmueller | lamersd] @informatik.uni-hamburg.de

Abstract

This papers argues that the *mobile agent* approach is well suited for sporadic communication in open distributed systems - especially for rather 'loose' cooperations across local and organizational borders: In an increasing number of cases, modern (workflow) management of distributed business procedures reaches beyond such borders. For most existing workflow management sytems this means that cooperations partner are required to give up their local autonomy. However, for cases in which business partners intend to cooperate but still need to preserve their local autonomy, process participation on the basis of mobile agents represents an attractive and appropriate mechanism.

Based on an order processing example, the paper shows how such kind of process integration can be achieved. It then demonstrates how the COSM (Common Open Service Market) system software can be extended in order to use petri net based process definitions with realize mobile agents in an integrated distributed system platform. Finally, basic design and implementation considerations of mobile agents in COSM are also described.

Keywords

Workflow management, mobile agents, electronic service markets, interorganizational communication

1 Introduction

Workflow management (WFM) is an enabling technology for the integration of process oriented tasks. It is best applicable in the setting of well-structured and well-informed organizations. However, WFM also reduces the local autonomy of organizational units involved in cooperative workflows. Whilst the required adaptation may be carried through *within* an organization, several coordination problems may arise *between* them:

- *Lack of a common communication infrastructure*: Separate companies may not be connected via a shared communication network - so they have to exchange data by telephone lines. They may operate on the basis of heterogeneous software environments w.r.t. operating systems, accounting software and even workflow management software itself. An integration of these components requires high set-up costs which reduce the benefit of interorganizational WFM.

- *Lack of central management*: On the market, participants coordinate their activities via 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 towards an integration as appropriate for the demander/ supplier relationship e.g. in the automotive industry field. Companies might rather intend to integrate isolated tasks from separated companies into their own processes. One may think of a manufacturer who wishes to initiate a quality assurance process in a supplier's production.

Here the costs of calling the involved 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.

- *High coordination costs of WFM systems*: Of course activities that aim to integrate interfaces of heterogeneous WFM software products have arisen in the past years [WFMC95]. But the result is an interoperability at a very high level that requires both partners to set up expensive equipment. Therefore, WFM software only pays in settings with highly repetitive activities. Efficiency would not lack if the level of interoperation is lowered to a more generic infrastructure allowing to integrate not only WFM applications but access to any other on-line services as well. In this case, the communication infrastructure is already given and the WFM integration costs decrease to the extension of such an existing infrastructure.

The mobile agent (MA) approach fits well to this WFM scenario whenever *ad-hoc communications* are to be addressed. As further elaborated in [MML96], a typical MA scenario is given by the *remote installation principle* where third-party vendors of mobile agents provide added-value services that facilitate customer access to remote information sources. This technique is, e.g., also well-known from the Java language which allows to download application components that might communicate with remote servers on behalf of their respective users [Sun95]. Programmed in this manner, Java applets provide a suitable abstraction towards the server's interface and behavior. In contrast to the Java approach, however, an MA *actively* migrates to a user's local host and utilizes its standardized API for data entry and general user interaction. The only software component that provides a well-known programming interface is such a *user agent platform*. The remote server which is to be visited by such an agent might be accessed through a proprietary interface (Fig. 1).

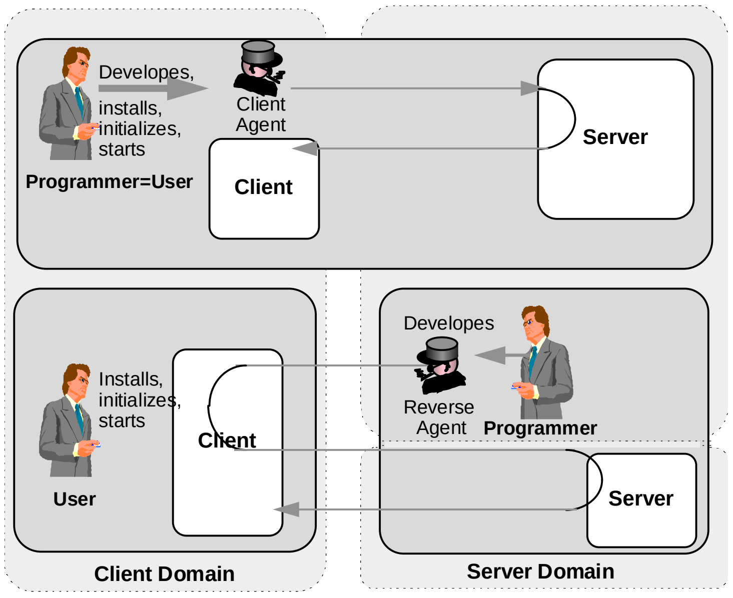

Fig. 1: Client agents vs. the remote installation principle

This paper proposes an architecture that applies the remote installation principle to WFM requirements. The MA infrastructure serves as a communication mechanism that bridges organizational boundaries. Therefore, Section 2 shows the specific problems of Interorganizational Work Flow Management (IOWFM) and Section 3 proposes the MA approach as an suitable solution. For the rest of the paper we assume a given electronic market infrastructure that provides a common platform to perform commercial activities, such as selling and purchasing services from an economic perspective, and to execute generally operations on remote servers from a technical view.

Such a common system software infrastructure for electronic markets is the goal of the COSM (Common Open Service Market) project [MML94a]. As presented in Section 4.1, the COSM infrastructure allows to perform remote operations at the server's side - for example in order to initiate immediate payment by the integration of an underlying payment services like, e.g., Ecash from DigiCash [Chau92]. It is based on a specific service description data structure - called ‘Service Representation’ (SR) - which is flexible and can be extended individually by any service provider in order to supply as much information as possible to the public (Section 4.3). Similar to WWW browsers, COSM provides a technical infrastructure that is easy to install on client hosts and thus doesn't involve participants with high set-up and operating costs. In this paper, we present an approach to interorganizational WFM that combines a generic communication platform with a flexible MA extension resulting into a new lean WFM system architecture.

2 Interorganizational Workflow Management

Workflow management deals with the specification and execution of business processes. General *process definitions* include *activities* to be performed, their control flow and data exchange, organizational roles of persons and software components that are allowed to perform activities, and policies that describe the organizational environment [WFMC95]. The objective of workflow management can be generally separated into the phases workflow process definition, WF application configuration, and WF execution.

What happens if external partners have to be involved into a process definition? It does not appear realistic to assume any knowledge about the interfaces of their respective local software components. The immediate integration of - maybe heterogeneous - WFM software does not seem feasible either. In the setting of a sporadic communication with autonomous trade partners the bottom line of common knowledge might thus be just the 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:

2.1 An Example Application Scenario

In a trade company, an order processing task is initiated by a customers request (step 1). As the first step, the stock server is invoked to check whether the item requested is available. If this is the case, an offer is made and returned to the client. If not, a list of potential suppliers is provided either by a clerk or by a software component. This process may involve several other input parts as well.

Each supplier, in turn, receives a dedicated offer request and returns its 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 FINANCING_REQUEST 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 for the customer.

This process comprises different kinds of tasks that are well-structured like the STOCK_INQUIERY function. They could be well performed by conventional client/server tools. Other tasks, for example the decision which supplier to involve, may rather 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 actual state of execution.

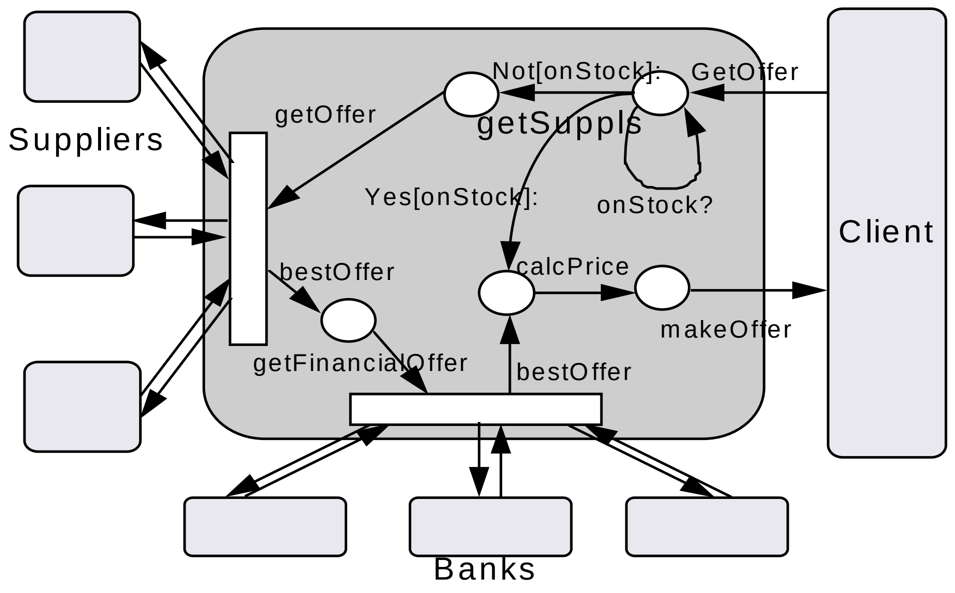

Fig. 2: An application process scenario

The data structure or *substrate* that represents a process instance may be a data structure local to any involved client or server or a mobile agent which carries this information around as a payload. In the latter case all information that is associated with the process instance is encapsulated as a single data object.

2.2 Relationship between Inter-organizational WFM and Agent Systems

Workflows are typically executed in distributed computing environments as, for example, in huge organizations or even in multi-organizational environments. Therefore, workflow management systems must be able to deal with both distribution and heterogeneity. Due to organizational modifications in the enterprise, workflow management systems have to be scaleable in order to satisfy changing requirements, e.g. when introducing new business processes or departments, or integrating new corporations or outsourcing existing services.

We assume that mobile agent systems will usually be utilized as the basis of electronic markets. Such markets are domains with

- customers and providers represented by software processes, and
- no dedicated providers and centralized management of the infrastructure.

Therefore, we can assume that agent systems must cope with the same problems that workflow systems have to deal with, i.e.:

- heterogeneous, dynamic, and distributed computing environments,
- high confidence and reliability requirements,
- high communication costs between applications involved, and
- flexible adaptation to new constellations of buyer-seller relationships.

For this reasons, mobile agents can appropriately be accommodated for the usage in workflow systems. In following, we claim that agent systems appear to be a suitable approach for building WFM systems that reduce the total set-up and transaction costs of such systems in the long run.

3 Mobile Agents

Work on mobile agents as a computer science concept reaches from distributed AI [Shoh93], distributed programming [Tsic87, HaCK95] to the field of computer communications [Tsch93]. This paper focuses on a client/server-oriented aspect of MA cooperation. A *mobile agent* is therefore defined as an encapsulation of *code, data*, and *execution context* which is able to *migrate autonomously* and *purposefully* within computer networks *during execution* [MML96].

The mobile agent system provides an algorithmically complete programming language environment. Therefore, an agent is able to react sophistically 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.

According to this definition, a migrating *process* is not an agent unless the process itself influences the migration target. A remotely executed *program* (see, e.g., Sun’s Java Applets [Sun95]) is not an agent either unless execution state is transferred and full execution autonomy is given. Agents only cease to exist when they are explicitly deleted. (Self-) deletion may be effected by an agent program as well. Agents - in this context - act on behalf of a (human) *principal*, who directly or indirectly defines the goal of an agent‘s activity.

The given definition restricts the possible variety of agent implementation approaches to a quite small set of possibilities: each case requires a local evaluator - the *engine* - in order to execute the agent program.

Among others, one important rationale for agent activity is the provision of *added value*: existing - maybe heterogeneous - resources, like booking services, are accessible through a customized or standardized user interface. In this case, a (commercial) third party provides the agent as a *facilitator service* in order to support access to "awkward" services. A *facilitator agent* can be transferred to the client‘s site and bound to the local interface library. This enables the users to access the facilitator service without using dedicated client software.

3.1 Mobile Agents as a WFM Infrastructure

As far as the example process is concerned, a MA may act as a facilitator for external parties to access data at the companies server or simply to transmit data entered by external partners. A supplier might be prompted to enter the price and delivery conditions via the agent’s user interface. Similarly, a bank clerk may offer credit conditions to the company in the given example of Section 2. Data representations and the agent program itself are supplied by the company without the need to negotiate interfaces and semantics of communicating software components. One of the main advantages of the MA approach becomes obvious in this context.

The MA approach thus satisfies already the lion’s share of IOWFM requirements:

- *Autonomy*: External partner’s engines may decide upon the agent and its local data which code to execute and which to reject. They may also implement a limited execution of untrusted agent code. The trade company, on the other hand, creates the agent autonomously as well, since negotiations on technical protocols are not required.
- Appropriate *communication* mechanisms: Generally, agent programming requires less communication bandwidth than synchronously communicating client/server applications due to the mobility feature of agents [Whit94].
- Appropriate *encapsulation of state and control*: The current (workflow) execution state is encapsulated by MAs as a run-time data value. Therefore, not only data that is communicated between individual activities involved in a process - also process execution is repre-

sented by the agent itself. As a data value, it can be saved to stable storage and thus consistency can be secured transactionally.

- *User interactions*: Agents are able to initiate a dialog with users and let them access local data. No application-specific API standardization is required.
- *Concurrency*: An agent might concurrently spawn several subagents in order to reduce overall process time. Depending on the agent platform, the spawned instances may later on merge their distinct results when joining in with the master agents.

Why are mobile agents an appropriate approach for WFM? It was stated in Section 2.1 that IOWFM requirements do match benefits of the MA approach. However, the existence of sophisticated agent platforms allows to adapt this technology without starting from the beginning. This leads to the following advantages:

- Encapsulation of state enables workflow instances to be saved always on stable storage. It simplifies the recovery process and thus enhances the overall fault tolerance of the system.
- The ability of migration allows to further decentralize a WFM system. This allows workflow participants generally to influence the scheduling of an agent by either modifying local data like locations to be visited or the agent program itself. The degree of intervention is controlled by the agent programmer. The more abstract the agent programming language, the easier may an agent be modified.
- Interpretation of agents provides homogeneity on an abstract machine level. This has two important effects: First, engines provide heterogeneity transparency as far as varying hardware and operating system architectures of IOWFM participants in different enterprises are concerned. Secondly, interpretation can maintain several security levels by excluding distinct sets of commands at the agent receiver's site. On the other hand, an agent provider, such as the trade company, may restrict other companies' access to local data resources by delivering an agent reduced to perform only a restricted task.
- Only one organization - the trade company - designs, implements and supplies the agent program. The agent's (and therefore the workflow's) functionality is realized by a single participant. This reduces coordination costs which may arise in case where different static programming interfaces between companies are involved.

Although mobile agent technology helps to coordinate local activities at separated organizations, they generally lack the ability to obey *role information* for the activities performed. Not every person may, for example, be permitted to calculate the price for a requested product or to select the appropriate bank that is to be requested for a credit offer. Therefore, role information still needs to be integrated into an agent program in order to satisfy authorization requirements.

Another requirement concerns the *traceability of process state*: How does a central control software determine the execution state of a distinct process if the respective agent migrates autonomously outside the company's domain? This question determines a general design option whether the coordination infrastructure of mobile agents should be centralized or decentralized. In the former case, a well-known server should be periodically contacted or visited by roaming agents in order to report the state of the workflow process they represent. Central information on the agent state is implicitly updated. In the latter case, agents only report on their execution state if explicitly programmed. This approach requires less infrastructural support.

3.2 The Trade Company, revisited

As a first refinement of the application example, it appears reasonable to take a closer look at it from the MA point of view. Two alternative approaches are applied to model the sample process either as autonomously migrating agents or as process instances represented by agents yet with a centralized control.

The *decentralized model* (Fig. 3) requires an infrastructure that allows to parameterize agents with migration targets: Supplier addresses are determined in the 2nd step in Fig. 3 and entered to the agent instance. As a next step, this single agent may spawn dedicated subagents in order to let each of them migrate to a respective supplier. The master agent then suspends its activity until a predefined time-out has been reached or all subagents have returned with individual offers or rejections. Those subagents which have not returned in time will be ignored if they (eventually) arrive. Although activities to be carried out by an agent are determined by the agent program, the hosts to be visited are not. Each single process instance may be represented by a distinct agent that, for example, visits different suppliers or banks.

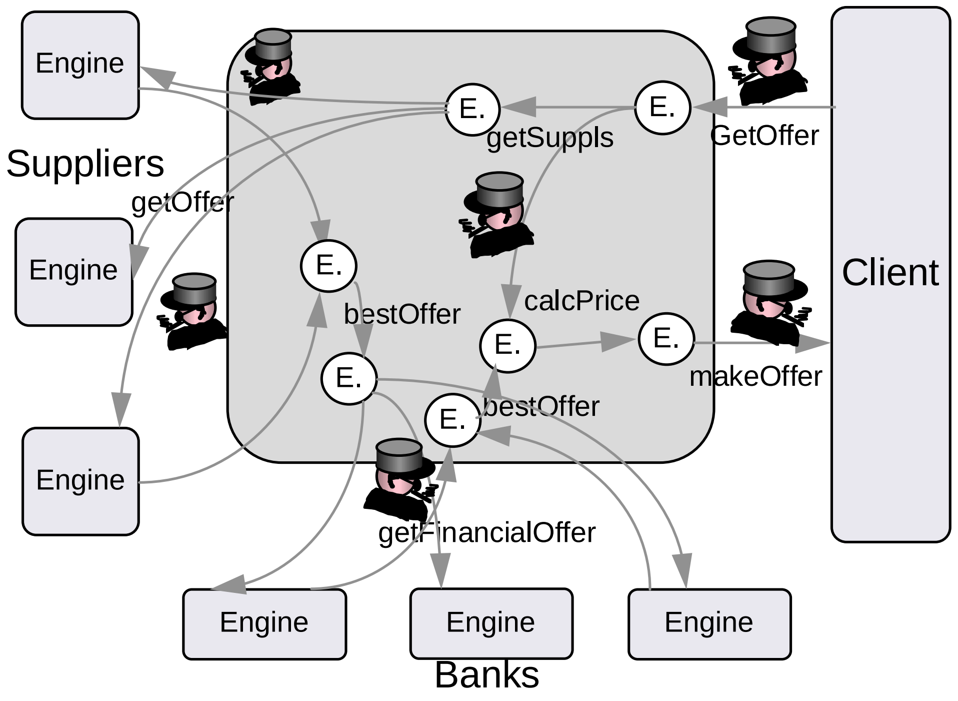

Fig. 3: Decentralized WFM with mobile agents

If the best offer from a supplier or a bank can be determined automatically or by involving a user depends on the agent program. Finally, the master agent has collected the appropriate information to make an offer for the initial client company.

In the *centralized model*, a *task server* maintains all agent instances that are involved in several process types. An agent is re-transmitted to the task server after an activity has been carried out. Workflow participants may acquire an agent from the task server on demand. An officer from the supplier’s side may thus place a query for agents that are ready to be processed. If there are agents that are in an appropriate execution state and the supplier has the right execution authorization, they are transmitted to the supplier’s engine.

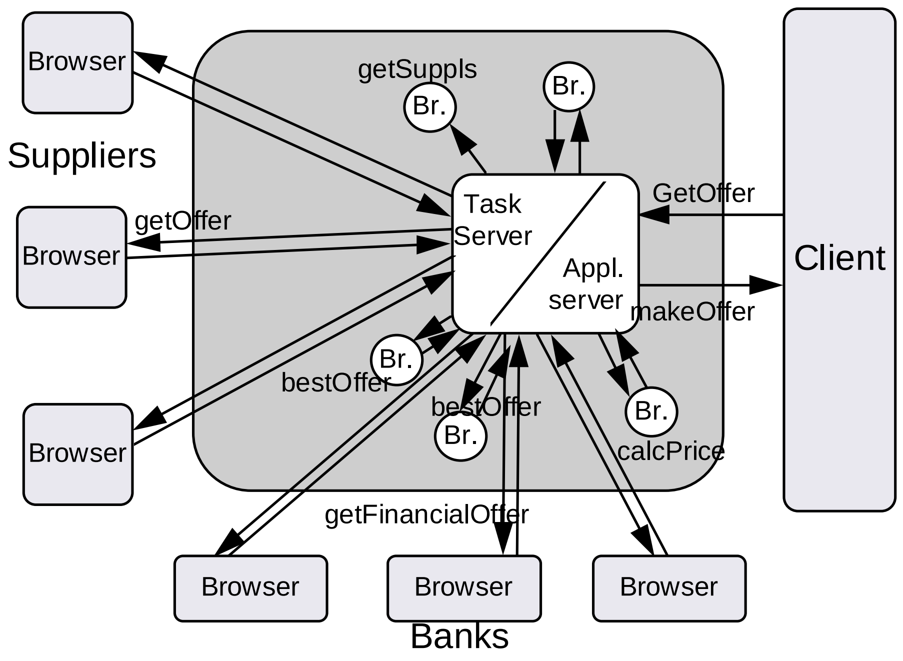

Fig. 4: Centralized WFM with a task server

This strictly centralized model does not necessarily require mobile agents to accomplish the respective task. In fact, a platform like Java might be involved to perform task of data entry outside the company's organization. In this model, an agent may just serve as a mediator between data entry and remote procedure invocations at the company's application server. However, *some* process activities may be carried out in a centralized way whilst others may rely on autonomous migration. In the application example above, external communications may be realized in a decentralized way and internal communication - which is independent of bandwidth restrictions - may also involve a centralized task server. Such a mixed model will be applied in the following - after introducing the COSM framework.

4 Mobile Agents in COSM

The COSM architecture was designed as an technical infrastructure to support communication between users and remote servers on electronic service markets [MML94a]. Such a service market requires flexible access to spontaneous appearing and developing remote services. The COSM infrastructure is a client/server platform that allows users to access remote services directly, i.e. without having to download dedicated client software in order to communicate with the server.

4.1 The COSM Architecture

The COSM system infrastructure allows service providers in open electronic service markets to offer newly created services without previous agreement on application interfaces between client and server developers. The specific advantage of the COSM architecture lies in the increased *design* and *execution autonomy* of service providers: Service providers are not obliged to adhere to existing service interface standards for their own operational interface.

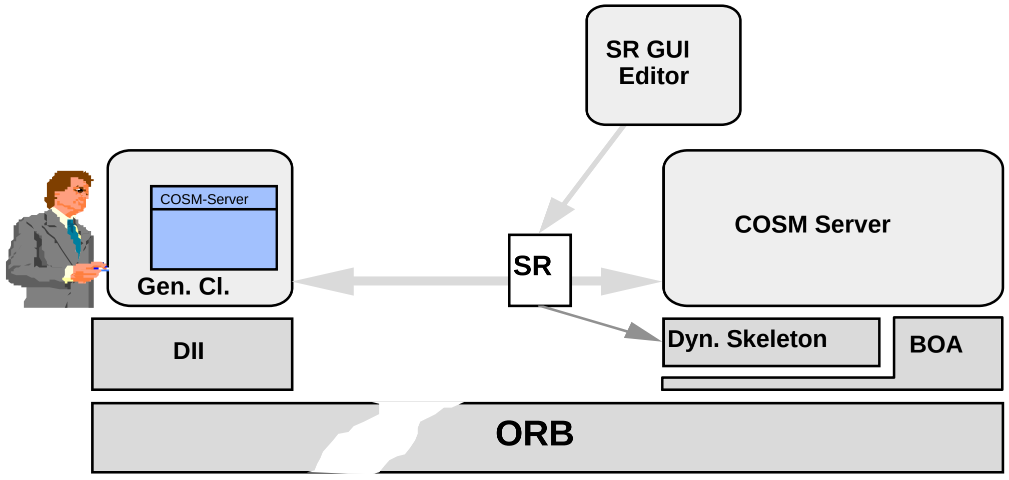

Fig. 5: COSM application model

The architectural model of COSM is based on the following main components (Fig. 5):

- A *Generic Client* (GC) component supports (client) users in service discovery, service access and binding, and in the inspection of service representations at run-time.
- A common *Service Representation* (SR) which contains description components required by the GC.
- *Service Providers* implement dedicated functionalities and are accessible on-line in the COSM network. They supply their respective SRs to their potential users (possibly via GC or *mediating services*).
- The overall architecture takes advantage of the a Common Object Request Broker Architecture (CORBA), which supports service access at distributed objects.
- Finally, *Mediation Services* provide application specific functions like, e.g., SR-*Browsers* and *Repositories*, which support users to inspect and store service representations.

In order to bind to a (usually unknown) remote server, a GC first receives its respective SR and then generates a corresponding user interface automatically. The user can inspect the information provided by the SR and - thus - may familiarize with the described service functionality. A session between GC and server might then be established (or released if this information does not describe the kind of service the user was looking for). In COSM, the user interface representation of a remote services site is standardized at the GC; layout and content, however, may vary from service to service. This interface enables users to execute remote operation calls just by filling out forms and pressing corresponding buttons automatically.

Service references are data values, which can be transferred across the network. On this basis, dedicated COSM services are enabled to act as directory services allowing servers to register their service representation and clients to obtain detailed information about the requested services. Such mediation services may act on behalf of their clients to evaluate the quality of services offered, or they may act on behalf of servers to „advertise“ registered entries.

In a commercial environment, interaction with remote servers frequently implies a *contract establishment* as well as a payment flow for service utilization. Therefore, current research also focusses on the integration of an anonymous electronic payment service [MML94b]. Accordingly, servers may *charge* their clients either on a connection basis, by operation invocation, or by the amount of data transferred.

4.2 Application Execution in COSM

The COSM *Generic Client* (GC) is a tool for human users to acquire remote server access and to support interactions between users and servers. Communication between client and server is based on a Dynamic Invocation Interface (DII) as defined by the OMG CORBA [OMG91] specification. The DII dynamic typing mechanism immediately results from ad-hoc binding requirements at the COSM-application level. Although late binding generally implies the possibility of a type mismatch at run-time, communication in COSM is type-safe, since RPC invocation parameter values are generated by means of SR operation signature descriptions. RPC data types correspond with user interface types in order to allow users to inspect and modify parameter and result values at the GC site. Besides standard DII data types, COSM provides special types for, e.g., service references, bulk data values, and ASCII files. By supporting service references as first-class data objects COSM servers are able to display directories of servers via the Generic Client to the user - similar to WWW hypertext references. Binding to a such referred server is effected directly from the user interface by selecting the according service reference. Loosely coupled chains of servers may emerge from this principle.

A comparison of COSM with, e.g., the *World Wide Web* (WWW) reveals some few but substantial differences between COSM and the WWW:

1. COSM allows immediate access to *software applications* with operational interfaces while WWW supports *document exchange*. However, interactive forms are supported by WWW, but such „operation calls“ only result in the delivery of a further document. Restrictions of WWW apply when results of previous operation calls are used as parameters for further ones (statefull servers).
2. The second difference concerns possibilities for *chaining* service providers to allow value chains to emerge at considerably lower costs. Since COSM servers present operational interfaces, they allow specific client software to automatically execute operation calls. This software does not provide a user interface, but may act itself as a COSM server. In addition, in order to let client calls directly pass-through to the back-end server, such front-end servers may carry out specific computations on parameter and result values. One such service is a dynamic transaction monitor that coordinates access to back-end services.
3. Compared to agent-oriented approaches or Java, SRs do not contain an executable program in the base infrastructure. However, since the SR is arbitrarily extensible program code, control flow descriptions may be introduced if required for dedicated application fields.

4.3 COSM Service Representation

The crucial data object to contain and transfer arbitrary service descriptions in COSM is the Service Representation (SR). At its most generic level, the SR is a container for data structures of arbitrary types at run-time. After receiving the SR, an SR-interpreter (as a component of the Generic Client) seeks descriptional components it is sensitive for. Among these components are:

- an *interface description*, containing operation names and *parameter descriptors*. These components resemble roughly interface definition clauses known from DCE or CORBA IDLs,
- a specification of the *user interface* to be generated for human users. It contains specifications for dialog boxes, data editors, and push buttons. Data editors manipulate local data objects of the SR.
- a specification of the service interface protocol specifying which operations are enabled to be invoked at a given state. Currently, this protocol description is based on a *finite state*

machine (FSM) model and comprises a set of application states and transitions between them which refer to operation descriptions. State changes are effected by user-level events and execute, in turn, RPC invocations.

- informal description components, e.g., help texts to support human users.
- „price tags“ that inform users on service access and related operation invocation fees,
- any application-specific data values required locally by the GC to maintain execution state.

It is possible to store the SR persistently and to suspend interactions with the remote server. This SR can be re-activated later - perhaps from another network site - in order to resume the previous communication state. This will prove to be an essential asset for the MA extension.

Two benefits arise from an SR-based approach to implement open service access: first, *representations* are standardized within COSM, not service interfaces. This includes a uniform, service-independent user interface presentation and allows type-safe data entry. Second, more application-specific functionality can be shifted to the client site. In fact, COSM servers are reduced to plain call libraries with corresponding SRs. Compared with, e.g., X-Windows, dialog control is shifted to the client part of a distributed application.

The Generic Client dialog management component connects presentation functions to the application call interface. Users are allowed to invoke remote operations by pressing corresponding buttons. In the case of statefull services, buttons are disabled if the operation is not allowed to be invoked in the current application state.

As containers for arbitrary data type definitions and value components, SRs are extensible at run-time. This allows specialized environments to introduce dedicated component types for their own purposes. COSM applications that are not aware of these extensions will still be allowed to interpret the subset of SR components they expect.

4.4 Mobile Agent Extension of COSM

One possible extension to service representation is the introduction of control flow definitions. As an ordinary data structure this control flow can be saved on stable storage and restored at a later time or on a remote host. Therefore several of the preconditions for a mobile agent platform are already given. How to represent the control flow, how to provide autonomous migration and how to integrate these with the existing parts of a service representation remain as open questions. In order to first achieve mobile agent capabilities and later on to extend it for IOWFM purposes the introduction of petri nets as a control flow representation will be discussed next:

4.4.1 Control Flow Model for Service Representation

Execution autonomy is an indispensable feature of an agent. It can be defined as the ability to decide upon its data which activity to perform next. In the mobile agent extension of COSM, control is carried out by a virtual machine (the *engine*)that resides on each host and decides by means of a petri net instance and agent's local data which command, if any, could be executed next. It is the local interpreter machine which controls execution of agents and which acts as an engine and the agent’s program that implement the particular execution unit.

Petri nets serve as a means for control flow description in workflow applications [Jens92]. An automata model evidently does not comply for this purpose since one of the requirements imposed on agent's capabilities is the option of a concurrent agent program execution. Besides, the petri net model provides the advantage to combine a graphical representation with a solid mathematical foundation as a formal description technique. The latter feature enables for example automatic determination whether there exists a deadlocks in a given net. The graphical representation in turn is easier understandable compared to a textual specification. This feature may allow even inexperienced users to create their own nets due to the simplicity of petri net syntax.

However, some modifications of the original petri net model have been introduced in its use in COSM SRs in order to accommodate them for agent control descriptions. First, where concurrency is modeled in an abstract control flow description, the concrete net representation of an agent is unfolded into a set of sequential net instances. Each of these instances controls a single agent's activity (Fig. 6).

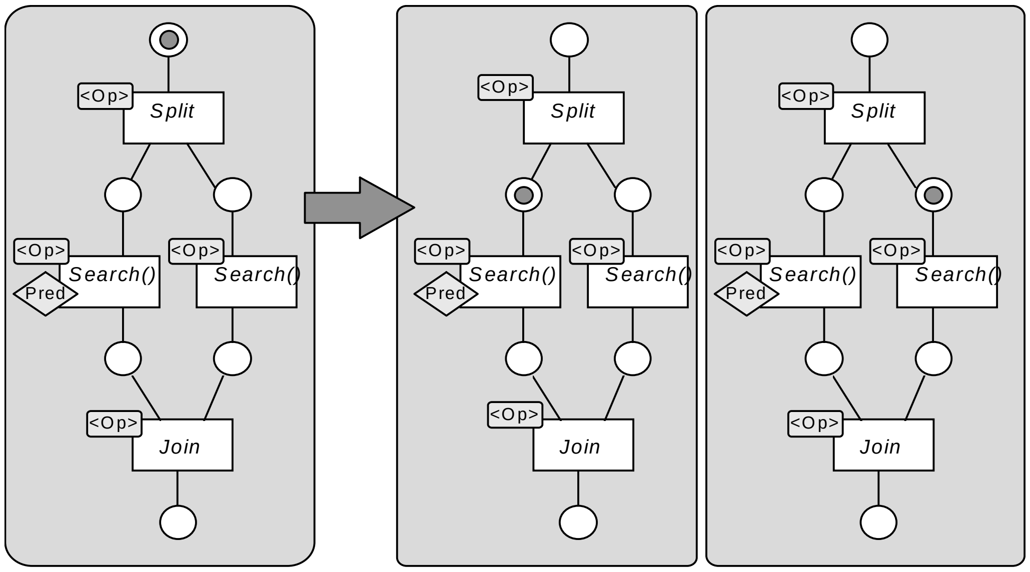

Fig. 6: Split- and join transitions

Concurrency is achieved by specific transition types in the net, which denote specific behavior: *split* and *join* transitions. Instead of creating several tokens in outgoing places after transition have been fired, several net instances of the same structure and state are created here. They differ, however, in their respective marking. Because concurrent processing applies most appropriately to distributed systems, emerging net instances are transferred to other hosts immediately after firing a split transition.

One of the net instances acts as a master that keeps track of the total number of created instances. In contrast to the split transition, a *join transition* gathers and collates all previously splitted instances as well as determines the master instance which will carry all information gathered by the others. Only these two types of transitions have more than one input or output place.

For a transition to fire, the input place must contain a token and the transition predicate must evaluate to "true". For the join transition to fire, all the instances created by the corresponding split transition must have arrived. Pairs of split and join transition can be nested. With each transition there is an operation definition associated. If the transition is fired the engine executes the specified function call.

An engine must be able on the other hand to determine what nets among those managed by it are to be merged while firing a certain join transition. It implies a dedicated naming schema that reflects the split-join relationship. The instances created while firing the split transition belong to a task that represents in turn an instance of a *task type*. One may construct an agent that requires a keyword to search for at a set of database servers. As soon as the task is instanciated, it has to be given a unique name in order to identify all net instances that belong to this task. Because the search agent may gather information at several places simultaneously, sub-agents can be sent to different information sources. Since all the instances must be named uniquely there is a universally unique identifier (UUID) element within each instance name. Therefore, the entire name is made up as follows:

- The *task type name* is chosen by the agent implementor.
- At each task creation, a UUID is appended to the original name as a *task identifier*.
- A unique *instance identifier* among all instances created by a certain split transition is appended and stripped off while firing either a split and or a join transition respectively.

This naming scheme ensures that the engine can recognize even among the nets of the same task which nets are involved in firing of a certain join transition.

4.4.2 Agent Migration

A dynamic communication technique applied to by the COSM infrastructure is COSM-DII, an implementation of the DII introduced as a part of the CORBA architecture [OMG91]. The advantage of the DII is its flexibility concerning data type representation at runtime. It is possible to build COSM-DII on the basis of standard CORBA-DII allowing the usage of Object Services and Implementation Repository.

Agents move as an effect of the "Go" command. When the engine encounters this command, it proceedes as follows: the entire agent will be stored by the means of SR-functions in a file. Then a transfer method of a remote engine will be called with this file as an argument. Other arguments could be supplied along with the file, for example authentication informations. The sending engine obtains the address of the remote one as a part of "GO" operation structure. After having arrived, the agent is passed to the engine by a gatekeeper process and execution will be continued from the command next after the "GO" command. After a successful transfer, the file on the departure host is removed.

4.5 Extending Mobile Agents for Workflow Applications

The mobile agent infrastructure of COSM lacks mainly the following extensions:

- first, information about the organizational role of the user who carries out an activity, such as price calculation,
- second, an extension to the generic client that allows users to log-in, and,
- third, the ability of the generic client to disable remote operation invocation by clicking a corresponding button in the user interface.

If, e.g., a bank clerk is prompted to enter a financing offer at his generic client all activities that are not allowed to be carried out by him must be prevented.

Further, the actual application server which maintains stock informations and allows participants to enter a price calculated on the one side, is separated from the *task server* which administrates service representations as process instances for a set of application servers. If a user wants to perform any task he is required to authentify himself at the task server and to transmit his role ID. The task server selects a to-do list consisting of all process instances with an execution state that allows this role owner to perform at least one activity. As a service representation the process instance is acquired by the generic client in order to let the user access the application server via remote procedure call. After this interaction has taken place, the SR is transferred back to the task server and might appear on the task list of another role owner.

If an SR is treated as an agent, it is immediately transferred to the respective engine. In the agent-based example of Fig. 3 the financial offer calculation is carried out by the bank's local engine. If this engine insists on invoking a remote operation it is not authorized to the application server is able to recognize this due to a missing or wrong authentication.

4.6 The Trade Company, revisited, revisited

After the COSM infrastructure and its MA extension have been introduced, it is appropriate to integrate these components.

- Our implementation scenario will therefore take advantage of the generic client in order to allow flexible access to remote servers,
- it will involve engines to execute agents automatically,
- a task server is involved where agents are kept as task list entries, and

service representations - as encapsulation of data and control state - will be treated as mobile agents if appropriate.

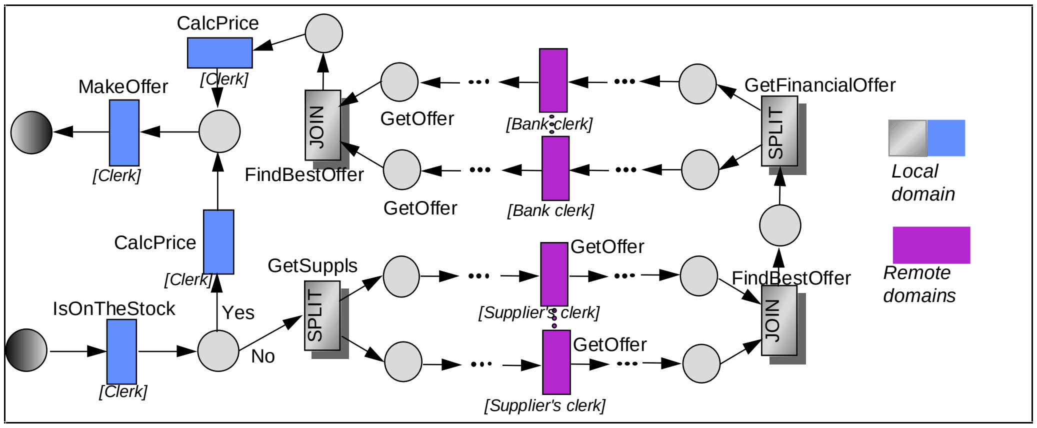

Fig. 7: Control flow specification

Fig. 7 shows a further refinement of the process definition from the previous example. Split and join transition are defined to control subagent creation and collection. Engines carry out this task and treat service representations as agents. Towards external business partners the task server acts as an engine that collects agents after an offer has been made and the respective agents migrated back to the local organization. *Within* this organization, the task server maintains all agents as service representations. A user authentifies himself at the generic client and requests the task server to enumerate all available service representations. The basic COSM approach to SR-based client/server interaction can be applied here.

The advantage of the interpretation of service representation - either in the 'traditional' way or as a mobile agent - yields in an additional flexibility in the process definition: It does not depend on the control flow description how an SR might be interpreted at run-time. Means of process definition and execution are therefore independent from one another. If the task server from the example of Fig. 4 is slightly modified in order to introduce engine capabilities, external participants may switch from the task-list approach to the agent based one. The SR will remain constant in both cases.

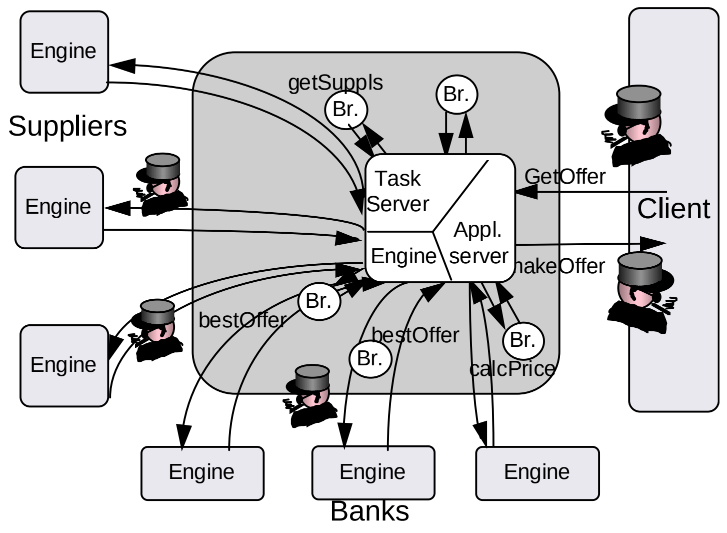

Fig. 8: Combining the centralized and the decentralized approach

The current prototype of the COSM mobile agent platform was developed in C++ for IBM AIX. A second implementation exists for IBM OS/2 with the option to run engines on both platforms and to exchange agents at a cross-platform level. The necessary heterogeneity transparency is achieved by a representation-independent implementation of the SR.

5 Summary and outlook

It was 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 applies 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.

Within the setting of the COSM project - which tackles generally cooperation support in the context of interorganizational client/server applications - an extension of the infrastructure towards mobile agent is provided and, in turn, IOWFM support was presented. The specific advantage of this approach lies in the encapsulation of agent execution state as well as application data within each service representation. Depending on the view of specific applications, such as the generic client or the engine, different aspects of the SR are of concern.

Having already obtained an encapsulation of state and process execution data, it is possible to store meta-information additionally in a service representation: transaction timestamps may help to trace each single activity step and to gather information on further process optimization possibilities. The COSM application model also provides a flexible security framework which involves payment and notary services [MML94b]. These could be involved into IOWFM applications first in order to log communications between trade partners in a non-repudiating way and, second, to integrate on-line payment services in order to utilize externally provided service, like, e.g., ordering a courier service.

The control flow representation itself may be a subject of user activities as well. If the process instance requires an exceptional treatment, the control flow can be changed at run-time. This should be supported by dedicated tools in order to maintain a consistent data structure. Such modifications may either concern single process instances or a process definition. In case of the latter, all future instances bear this modification.

6 References

[Chau92] D. Chaum: *Achieving Electronic Privacy*, in: Scientific American, Aug. 1992, pp. 96-101

[HaCK95] C. Harrison, D. Chess, A. Kershenbaum: „Mobile Agents: Are they a good idea?“, IBM Research Report #RC 19887, 1995

[Jens92] K. Jensen: „Coloured Petri Nets“, Vol. 1, Berlin Heidelberg New York, 1992

[MML94a] M. Merz, K. Müller, W. Lamersdorf: "Service Trading and Mediation in Distributed Computing Systems", Proc. IEEE International Conference on Distributed Computing Systems (ICDCS), L. Svobodova (Ed.), IEEE Computer Society Press, 1994, pp. 450-457, Los Alamitos 1994, pp. 450-457

[MML94b] M. Merz, K. Müller, W. Lamersdorf: "Trusted Third-Party Services in COSM", in 'EM - Electronic Markets', Institute for Information Management, Universität St. Gallen, Schweiz, Heft 12, September 1994

[MML96] M. Merz, K. Müller, W. Lamersdorf : "Agent, Services, and Electronic Markets - How do they integrate?“, in: A. Schill, O. Spaniol (eds.) Proc. ICDP International Conference on Distributed Platforms

[OMG91] OMG: "The Object Request Broker: Architecture and Specification": OMG Document No. 91. 12. 1, Object Management Group, Framingham, MA, USA, 1991

[Shoh93] Y. Shoham: „Agent-oriented programming“, in: Artificial Intelligence, Vol. 60(1993), pp. 51-92

[Sun95] Sun Microsystems: HotJava, http://www.javasoft.com, White Paper, 1995

[Tsch93] C. F. Tschudin: „On the Structuring of Computer Communications“, PhD thesis, University of Geneve, 1993

[Tsic87] D. Tsichritzis, E. Fiume, S. Gibbs, O. Nierstrasz: KNOs: Knowledge Acquisition, Dissemination, and Manipulation Objects, in: ACM Transaction on Office Information Systems, Vol. 5, No. 1, Jan. 1987

[WFMC95] Workflow Management Coalition, Reference Model and API specification, http://www.aiai.ed.a.c.uk/WfMC

[Whit94] J.E: White: "Telescript Technology: The Foundation for the Electronic Marketplace", White Paper, General Magic Inc., 1994