Inter-Enterprise Collaborative Business Process Management

Qiming Chen, Meichun Hsu
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-107
August 17th, 2000*

E-mail qchen@hpl.hp.com

cooperative process management

Conventional workflow systems are primarily designed for *intra-enterprise* process management, and they are hardly used to handle processes with tasks and data separated by enterprise boundaries, for reasons such as security, privacy, sharability, firewalls, etc. Further the cooperation of multiple enterprises is often based on peer-to-peer interactions rather than centralized coordination. As a result, the conventional centralized process management architecture does not fit into the picture of inter-enterprise business-to-business E-Commerce.

We have developed a *Collaborative Process Manager* (CPM) to support decentralized, peer-to-peer process management for *inter-enterprise* collaboration at the business process level. A collaborative process is not handled by a centralized workflow engine, but by multiple CPMs, each represents a *player* in the business process. Each CPM is used to schedule, dispatch and control the tasks of the process that the player is responsible for, and the CPMs interoperate through an inter-CPM messaging protocol. An XML based *Collaborative Process Definition Language*, CPDL, extending the process definition language (PDL), is developed for specifying collaborative business processes. We have implemented CPM and embedded it into a dynamic software agent architecture, E-Carry, that we developed at HP Labs, to elevate multi-agent cooperation from the conversation level to the process level for mediating E-Commerce applications. We have also integrated E-Carry with E-Speak, an inter-enterprise communication infrastructure product developed at HP.

In general, our approach represents a shift from centralized process management to decentralized, collaborative process management. We believe that CPMs will be the basic building blocks for a scalable, dynamic, inter-enterprise middleware framework. The feasibility and practical value of this approach have been demonstrated by the prototypes implemented at HP Labs.

Inter-Enterprise Collaborative Business Process Management

Qiming Chen and Meichun Hsu

HP Labs
Hewlett Packard Co.
1501 Page Mill Road, MS 1U4
Palo Alto, California, CA 94303, USA
+1-650-857-3060
qchen@hpl.hp.com

Abstract

Conventional workflow systems are primarily designed for *intra-enterprise* process management, and they are hardly used to handle processes with tasks and data separated by enterprise boundaries, for reasons such as security, privacy, sharability, firewalls, etc. Further the cooperation of multiple enterprises is often based on peer-to-peer interactions rather than centralized coordination. As a result, the conventional centralized process management architecture does not fit into the picture of inter-enterprise business-to-business E-Commerce.

We have developed a *Collaborative Process Manager* (CPM) to support decentralized, peer-to-peer process management for *inter-enterprise* collaboration at the business process level. A collaborative process is not handled by a centralized workflow engine, but by multiple CPMs, each represents a *player* in the business process. Each CPM is used to schedule, dispatch and control the tasks of the process that the player is responsible for, and the CPMs interoperate through an inter-CPM messaging protocol. An XML based *Collaborative Process Definition Language, CPDL,* extending the process definition language (PDL), is developed for specifying collaborative business processes. We have implemented CPM and embedded it into a dynamic software agent architecture, E-Carry, that we developed at HP Labs, to elevate multi-agent cooperation from the conversation level to the process level for mediating E-Commerce applications. We have also integrated E-Carry with E-Speak, an inter-enterprise communication infrastructure product developed at HP.

In general, our approach represents a shift from centralized process management to decentralized, collaborative process management. We believe that CPMs will be the basic building blocks for a scalable, dynamic, inter-enterprise middleware framework. The feasibility and practical value of this approach have been demonstrated by the prototypes implemented at HP Labs.

1. Introduction

E-Commerce applications operate in a distributed environment involving multiple parties with dynamic availability, and a large number of heterogeneous information sources with evolving contents. A business partnership is often created dynamically and maintained only for the required duration such as a single transaction. E-commerce activities typically rely on business-to-business (B2B) and business-to-consumer (B2C) interoperation at the business process level.

The automation of these activities represents both challenges and opportunities for supporting *inter-enterprise business process management.*

1.1 The Problem

The general function of a workflow engine is to support the modeling and execution of business processes [34]. Although the tasks that contribute to a process can be distributed, they are centrally scheduled at the process level. Such centralized process control is appropriate for a single enterprise. However, *intra-enterprise* process management and *inter-enterprise* process management are significantly different. When multiple parties belonging to different enterprises, are involved in a business process, they are unlikely to use a centralized process management, because they are often separated by firewalls, have self-interests, and do not wish to share all the process data. Rather, they need support for peer-to-peer interactions. This has become the major impendence for using the conventional centralized workflow systems for inter-enterprise E-Commerce automation. In fact, to our knowledge, there has been no such experience reported.

1.2 The Solution

Our solution to the above problem is based on extending process management from the one-server model to the multi-server peer-to-peer model, a shift from *centralized process management* to *collaborative process management.*

We introduce the notion of a ***collaborative business process*** (Figure 1). A collaborative process involves multiple parties. The process definition is based on a commonly agreed operational protocol, such as the protocol for on-line purchase or auction. A collaborative process is not executed by a centralized workflow engine, but by multiple engines collaboratively. More exactly, each execution of a collaborative process, or a *logical process instance*, consists of a set of *peer process instances* run by the Collaborative Process Managers (CPMs) of participating parties. These peer instances share the same process definition, but may have private process data and sub-processes. The CPMs run these peer instances independently and collaboratively. The CPM of each party is used to schedule, dispatch and control the tasks that party is responsible for, and the CPMs interoperate through an inter-CPM messaging protocol to synchronize their progress in process execution. An XML[2]-based Collaborative Process Definition Language, CPDL, extending the process definition language (PDL), is developed for specifying collaborative business processes. Solutions for synchronizing collaborative process execution are developed.

For example, in case a buyer wants to buy something from a seller, the buyer-side CPM engine, *A*, creates a logical instance of the purchasing process, and initiates a “buyer-side” peer instance; *A* then notifies the seller-side CPM, *B*, to instantiate a “seller-side” peer instance of the purchase process. The peer process instances at both sides can be considered as autonomous but are following a purchase protocol both the buyer and the seller are willing to comply. When *A* finishes a task, it informs *B* of the task status, in order for *B* to proceed, and vice versa. The entire purchase process is not handled by any common server, but by the peer-to-peer cooperation of multiple servers.

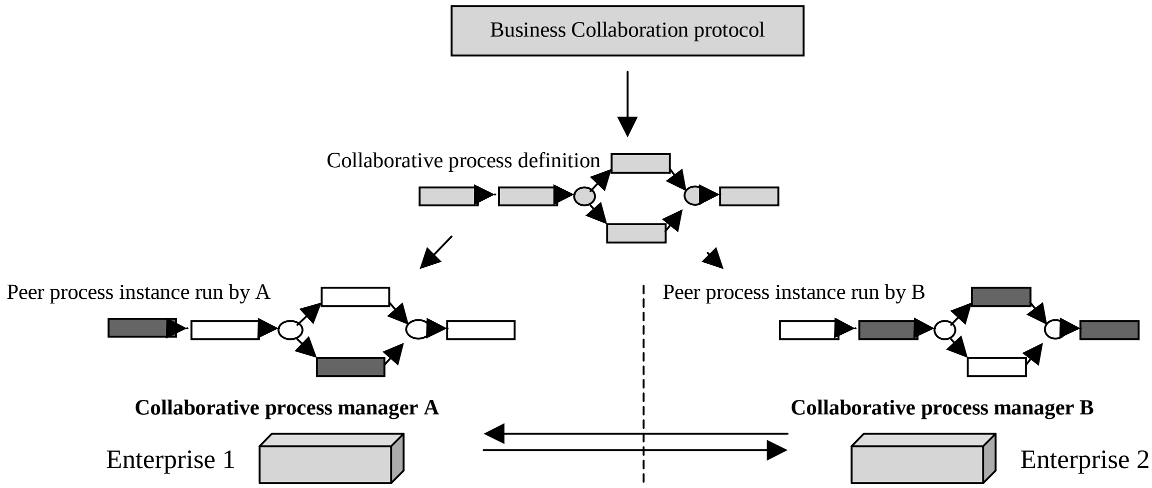

Figure 1: *Peer-to-peer collaborative process management*

Further, we integrate collaborative process management with an agent infrastructure, E-Carry, that we have developed at HP Labs. We show how agent-embedded CPMs can be used to shift agent cooperation from the agent conversation level to the process level, while at the same time shifting workflow management from centralized process management to collaborative process management. We have developed prototypes at HP Labs to illustrate the feasibility and practical value of the proposed approaches for enabling agent-mediated E-Commerce.

We claim that the proposed collaborative process management can provide a significant extension to the current workflow technology. It enhances the interaction of dynamically formed business partnerships, allows us to support inter-enterprise business cooperation at the process level, and represents a step towards a dynamic distributed middleware infrastructure.

Section 2 compares collaborative process management with other workflow schemes. In Section 3 the collaborative process model is described. Section 4 discusses the execution issues of collaborative processes. Section 5 describes the integration of CPM with an agent architecture, and illustrates the use of CPMs to support multi-agent cooperation. Some concluding remarks are given in Section 6.

2. From Centralized Process Management to Collaborative Process Management

2.1 Centralized Process Management

Workflow servers are used to coordinate the execution of multiple actions that form a business process [13,14,25,34]. A business process specifies the integration and synchronization of multiple steps, each step represents a logical piece of work, or action, that contributes to the accomplishment of the whole process. Although these actions and the agents executing the actions can be distributed, they are scheduled and coordinated by a centralized workflow engine. Typically a business process includes a *data packet* containing the *process data* for flow control and data flow, and tasks can manipulate the process state by updating these data. However, as shown in Figure 2, consider a purchase process involving tasks belonging to different enterprises, e.g. the buyer and the seller. It is unrealistic to have the buyer and the seller coordinated by a single workflow engine, and it is unreasonable for them to put their private data (e.g. negotiation threshold) into the common process data packet for flow control.

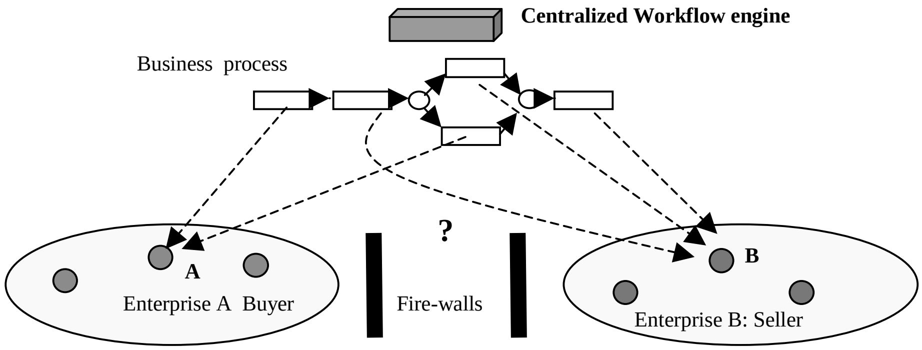

Figure 2: *Centralized workflow control: not for cross-enterprise applications*

2.2 Subprocess Execution

A task belonging to a business process, say *P*, may itself be a process, *P'*, referred to as a subprocess of *P* [34] (Figure 3). When *P'* is bound to *P* at the process definition phase, *P'* becomes a static extension of *P*. When *P'* is bound to *P* at the process execution phase, *P'* becomes a dynamic extension of *P*. In any case *P'* inherits some property, including the definitions (templates) of the process data, from *P*, while having its own specialized properties and data. As *P'* is just the extension of *P*, they are typically executed in the same enterprise.

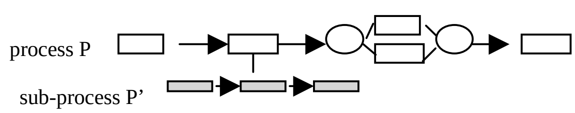

Figure 3: *A sub-process executed in the same enterprise as the parent process*

2.3 Multi-processes Interoperation, or Federation

Multiple individual business processes may be executed concurrently but with interoperability. For example, two processes, say P_1 and P_2, may interoperate in the following ways.

- Some tasks of P_1 and P_2 have operational dependencies. For example, task T_i of P_1 depends on the termination of task T_j of P_2 to start, such that T_i cannot start until T_j terminates (Figure 4).
- P_1 and P_2 exchange data at certain steps.

The Workflow Coalition (WfC) published recommended interface specifications for process interoperation.

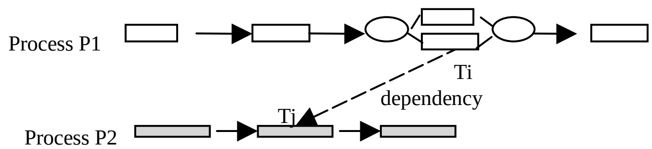

Figure 4: *Inter-operation of different processes*

It is worth noting the following features of the conventional process interoperation, in order to distinguish it from our proposed collaborative process management.

- The conventional process interoperation, or federation, primarily focuses on intra-enterprise applications. It lacks support for inter-enterprise cooperation.
- The conventional notion of process interoperation deals with the relationships between different processes. Although these processes may run on the same or different workflow engines, each process is fully executed by that engine. For each individual process, besides certain dependencies with others, the whole flow control is based on its own logic and execution progress.

2.4 Transaction Group

Several advanced models on transaction groups and cooperative transactions were reported in [3,7-11,19,21,28,32,34]. These models are characterized by joint execution of a transaction by multiple participants, and applied to such applications as cooperative design. The obvious difference of our approach from the above efforts is that we tackle the peer-to-peer interaction where no joint task is involved. We focus on inter-enterprise applications where participating parties, such as a buyer and a seller, deal with each other but have their own private database and decision rules.

2.5 Partner Interface Process (RosettaNet)

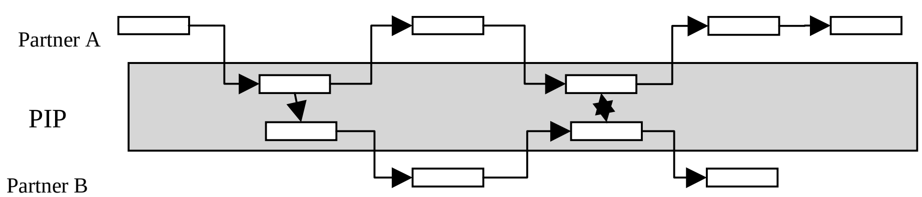

Figure 5: *Partner Interface Process (Rosetta-net)*

The RosettaNet Consortium, founded in 1998, has placed focus on defining standard interfaces between partners for business process integration [31]. More specifically, the consortium is driving the development of Partner Interface Processes (PIPs) that define the processes and data elements necessary for a broad set of supply chain scenarios. The PIPs only define the “interface” tasks that supply chain partners commonly participate in, but not the internal, proprietary processes used by any partner to carry out businesses. It is the responsibility of each partner to identify how its internal processes and systems align to the PIPs. This concept is shown in Figure 5.

The PIP approach does address the issue of inter-enterprise process integration for enabling plug-and-play for new partners into the supply chain. However, the PIP specifications focus primarily on architecting the information to be exchanged at the connection points of partner business processes; they do not focus on a common process-level specification for all the partners. Further, the PIPs do not offer a model of execution; for instance, it does not intend to specify how the partner process instances are synchronized, or made to be aware of the progress of the peer processes. The CPM approach discussed in this paper can be used to support PIPs. We can convert PIPs into process definitions in CPM, and support their execution.

2.6 Peer-to-Peer Collaborative Processes

The proposed peer-to-peer collaborative process management is different from all the above approaches. With this approach, an inter-enterprise business process is offered a global view, but executed by multiple distributed CPMs of the participating parties. An inter-enterprise collaborative process is defined based on the corresponding business protocols, and such a definition becomes the common template for all the participating parties to share. However, an execution of a collaborative process, viewed as a logical instance of the process, actually includes multiple peer instances that are not executed by a centralized workflow engine but by multiple CPMs and synchronized through peer-to-peer communication. The CPM at each side recognizes its own share of the tasks (*shaded* in Figure 6) based on role-matching. For example, an on-line trading process, say *P*, is executed collaboratively by a seller and a buyer in such a way that each peer CPM runs an individual process instance of *P*. For the CPM at buyer side, it is only responsible for (schedule and dispatch) the tasks to be executed by the buyer, such as preparing a purchase order and making a payment. Similarly the CPM at seller side is only responsible for the tasks belonging to the seller. The CPMs exchange task execution status messages for synchronization.

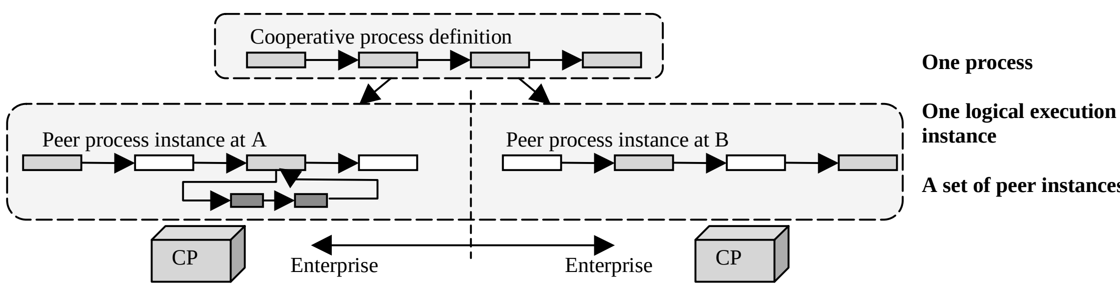

Figure 6: *Logical view to the execution of an inter-enterprise collaborative process*

Compared with the conventional workflow and sub-process handling techniques, this approach differs in that it uses decentralized and collaborative process management. Note that here the decentralization is introduced to the process management level rather than the task execution level.

The conventional process interoperation, or federation, approach, which supports task dependency enforcement and process data exchange, does not address many inter-enterprise cooperation issues. Also, the concurrently executed process instances do not follow the same process definition based on commonly agreed business protocols. Compared with that, the proposed collaborative process management has a clear focus and systematic support on protocol based inter-enterprise process management.

We share the same motivation as RosettaNet PIP approach in inter-enterprise process integration, and we conclude that our approach is capable of supporting PIPs. However, our approach goes beyond PIP in the following aspects.

- First, collaborative process management is based on **process-level business protocols**. Given a collaborative process, *P*, although each party is only responsible for a few steps of *P*, it can have a global view to the whole business process from the shared process definition. On the other hand, the PIP approach is **interface based**. PIPs expose individual “hand-shake” or conversation points of partner processes, but not a process level view to their cooperation. The PIP approach can be more appropriately viewed as a design at the conversation level than at the process-level.

- Second, we have developed peer-to-peer execution mechanism for collaborative processes. As mentioned above, each execution of a collaborative process consists of logically related process instances, for which we provide an execution model. In the PIP approach, however, there is no corresponding execution model. The execution of partner processes are not related and synchronized at process-level. Each party sees the trees, not the forest

From the above comparison we can see the uniqueness of the proposed approach in supporting peer-to-peer collaborative processes.

3. Collaborative Process Definition

To explain how the proposed collaborative process management approach extends the current workflow technology, we adopt the usual concepts of business process modeling in the following discussions. A *process* is modeled as a DAG with nodes representing the steps, or tasks, and arcs representing the links of those steps. A *work-node* represents a step (*task*) and associated with an activity, i.e. a piece of work that contributes to the accomplishment of the process, that may be executed either by a program (e.g. a software agent) or by a human worker. A process is associated with a packet of *process data*. When an activity is launched, a subset of the process data, sub-packet, is passed to it; when it is completed, together with task status information, the sub-packet, possibly updated during the task execution, is sent back for reconciliation with the process data packet. A *route-node* specifies the rules and conditions for flow control, process data update, etc. Conventionally, a process execution creates a single process instance. However, for a collaborative process, the logical instance of each execution includes multiple peer process instances. Further, a collaborative process may have multiple concurrent executions.

To support collaborative processes, the minimal extensions to process definition include the following.

- A collaborative process has a list of **process-roles**, indicating the logical participants. For example, if a simple purchase process has two roles, “buyer” and “seller”, then there are two peer instances involved in its execution, one at the CPM for “buyer” and another at the CPM for “seller”. These two peer instances are assigned roles “buyer” and “seller” respectively.
- A work-node has a **task-role**, and that must match one of the process-roles. In the above example, tasks can have roles “buyer” and “seller”. If the role of a task is “buyer”, it is only executed in the peer process instance with process-role “buyer”.
  Note that an activity also has a role called **activity-role**, such as “invoice-generator”, meaning that this task should be executed by (or dispatched to) an agent playing the “invoice-generator” role in this process. The notion of activity-role can be found in regular business process specifications.
- In an inter-enterprise collaborative process execution, each party wants to keep some of the process data prviate. For example, the buyer in one enterprise and the seller in another enterprise do not want to expose their thresholds during price negotiation. In the process definition, **templates** for holding the definitions and initial values of process data objects can be specified. Furthermore, the **sharing scope** of the data objects is specified. A template may be *public*, i.e. sharable by all process-roles (and thus by all peer process instances) or *process-role specific*. A role-specific template is used by the peer process instances of the given roles (one or more) only, and such templates can be made different for different process-roles. Consider a collaborative process with roles “buyer”, “seller” and “bank”; some data are private to “buyer”; some are sharable by “buyer” and “seller”; some are public to all three roles. The initial data packet of a peer process instance consists of the appropriate templates, where the sharing scope of each data object is marked. This data packet can be updated or expanded at run time.[1]

Let us use a simple example for explanation purpose. The sample collaborative process for on-line purchase defined based on the OBI (Open Buying on Internet) protocol, *obi_process*, has process-roles “buyer” and “seller”. Each logical instance of *obi_process* has two peer-instances run at two peer CPMs, *A* and *B*, one at the buyer side and one at the seller side. It has several tasks (steps) including T_1 (make purchase order), T_2 (process purchase order), etc. T_1 is a step the buyer is responsible for, so its role is “buyer”, while the role of T_2 is “seller”. *A*, running the peer instance with role “buyer”, is responsible for excuting T_1, and *B*, running the peer instance with the role “seller”, is responsible for executing T_2. The initial data packet for process-role “buyer” includes templates *obi_tpl* and *obi_buyer_tpl*, while the initial data packet for process-role “seller” includes template *obi_tpl* and *obi_seller_tpl*. The activity “*Action2* has an activity role “order examiner”, and thus it is dispatched to an agent with activity-role “order examiner” for execution.

[1] In this paper we do not explicitly address the situation where a single process role is played by multiple players (as in an example where multiple sellers coexist in a buying process). Such a situation requires extensions to both process definition and process execution described in this paper.

The specification of this process is illustrated below. It is WFC (Workflow Coalition [34]) standard compliant but is in XML format. When compiled, it is first translated into a DOM (Document Object Model) [15] tree of Java objects, then into a Java class for cooperative process definition.

```
<PROCESS name="OBI_PROCESS" ...>
        <ROLES> Seller, Buyer </ROLES>
        ...
        <WORK_NODE name="T1">
                <ROLE> Buyer </ROLE>
                <DESC> Make PurchaseOrder </DESC>
                <ACTIVITY> Action1 </ACTIVITY>
        </WORK_NODE>

        <WORK_NODE name="T2">
                <ROLE> Seller </ROLE>
                <DESC> Propcess PurchaseOrder </DESC>
                <ACTIVITY> Action2 </ACTIVITY>
        </WORK_NODE>
        ...
        <ARC name="Arc0" type="START"> <FROM></FROM> <TO>WorkNode1</TO> </ARC>
        <ARC name="Arc1" type="FORWARD"> <FROM>WorkNode1</FROM> <TO>WorkNode2</TO> </ARC>
        ...
        <PROCESS_DATA>
                <TEMPLATE> obi_tpl</TEMPLATE>
        </PROCESS_DATA>

        <PROCESS_DATA>
                <ROLE> Seller </ROLE>
                <TEMPLATE> obi_seller_tpl</TEMPLATE>
        </PROCESS_DATA>

        <PROCESS_DATA>
                <ROLE> Buyer </ROLE>
                <TEMPLATE> obi_buyer_tpl</TEMPLATE>
        </PROCESS_DATA>
</PROCESS>

<TEMPLATE name="obi_seller_tpl"> ... </TEMPLATE>
<TEMPLATE name="obi_buyer_tpl"> ... </TEMPLATE>
…
<ACTIVITY name="Action2" type="PROCESS" imp="AGENT">
        <DESC> Process purchase order </DESC>
        <ROLE> order examiner </ROLE>
        <CLASS> PurchaseOrderResult</CLASS>
        <URL> file:cba.hp.com/ecarry/CBLclasses </URL>
        <ARGS> ... </ARGS>
</ACTIVITY>
```

A task may represent a private sub-process with a private data packet. The sub-process binding is dynamic, that is, bound at run time. This allows a private sub-process to be designed separately from the host process (Figure 7). Furthermore, the process data of the internal sub-process is entirely private to the party executing the sub-process. Below is an example.

```
<ACTIVITY name="Action7" type="PROCESS" imp="SUBPROC">
        <DESC> Check credit </DESC>
        <SUBPROC> Check_credit_process </SUBPROC>
</ACTIVITY>
```

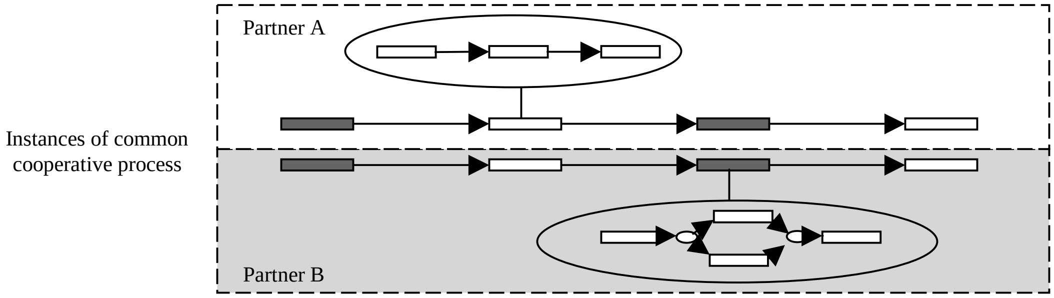

Figure 7: *Handle enterprise internal activities and data by private subprocess*

4. Collaborative Process Execution

An execution of a collaborative process consists of a set of peer process instances run by the CPMs of the participating parties. These instances share the same process definition but they have additional properties and may have private process data and sub-processes.

4.1 Collaborative Execution

Each peer process instance has a role that must match one of the process-roles. When a peer process instance is launched by a CPM at the seller side, for example, the process-instance-role is "seller", and the CPM is only responsible for scheduling and dispatching the tasks with task-role "seller".

When executing a collaborative business process, the *player* of each peer process instance must be specified and bound to the corresponding process instance role. In addition, a *logical identifier* for this execution must be obtained. These two pieces of information are captured as properties in every peer process instance. They are described below.

- **Players**: this indicates the CPMs participating in the execution of a collaborative business process. A player is associated with four attributes.
  - The role, e.g. "buyer" or "seller", of the given process instance running at the CPM that represents this player. Note that without binding to a peer process-instance, a CPM does not have a fixed role.
  - The domain name; a domain is a group of communicating servers coordinated by a *coordinator server* of that domain. The name of the domain is the name that the coordinator uses to register with an inter-domain messaging service infrastructure, such as HP E-Speak [16]; the coordinator can be thought of as the *gateway* to the domain; an example of a domain name can be "corp.hp.com".
  - The local name of the CPM server within the domain to represent the player. Each server has a unique local name within a domain. While a domain may have multiple CPM servers, one or more CPMs are selected to represent the players in this process instance. For example, *corp.hp.com/buying_agnet* may be a player playing the *buyer* role in a *purchasing* business process, whose peer CPM in this process might be *us.oracle.com/sales_agent.*

- The inter-domain messaging service infrastructure, such as HP E-Speak, that provides messaging services for inter-domain CPM communication. The messaging service infrastructure is capable of delivering messages among multiple domains. When inter-domain CPMs rely on E-Speak to reach each other, the addressing structure is
  espeak:domain_name/local_name.
  An example is *espeak:corp.hp.com/buying_agent*. More detailed message delivery mechanism will be explained later.

- **Coop-key**: this is used to identify a logical instance of a collaborative process, that is, to correlate and synchronize the multiple peer instances of the execution of a single collaborative process. All the messages exchanged for that execution are marked by a unique coop-key. In our implementation, each CPM can run multiple process instances concurrently, and each instance has a local ID. Each CPM engine maintains a mapping table between coop-keys and local process instance IDs. When a message relating to the execution of a collaborative process is received, the coop-key is used to identify the corresponding local process instance.

As shown in Figure 8, when a collaborative process is *defined*, it is specified with the process-roles and task-roles. When a logical process instance is *created*, the players and the roles they play are specified. The CPM at the creating party obtains a coop-key for this logical process, creates a peer process instance for itself, and associates this key with its peer process instance. When the CPM at the creating party sends requests to other peer CPMs (i.e., the other players of the process) to *instantiate* the peer process instances, the coop-key is also specified. This coop-key is encapsulated in all the messages on the above logical process instance, and transferred to all peer sides to correlate peer instances of the collaborative process execution.

Define Process

Roles	Buyer, Seller
Taskrole	T1/Buyer, T2/Seller
Proc_instant role	
Plyers	
Key	
Proc data packet	

Create Process Instance

Roles	Buyer, Seller
Taskrole	T1/Buyer, T2/Seller
Proc_instant role	Seller
Plyers	Seller|D1/A, Buyer|D2/B
Key	
Proc data packet	Data instances

Instantiate Peer Process Instances

Roles	Buyer, Seller
Taskrole	T1/Buyer, T2/Seller
Proc_instant role	Seller
Plyers	Seller|D1/A, Buyer|D2/B
Key	D1/A-100001
Proc data packet	Data instances

Figure 8: *Settings for defining, creating and initiating collaborative process*

Using the above *obi_process* as an example, the execution of a collaborative process is carried out in the following way:

- CPM *A*, representing the process-instance-role of "buyer", initiates a buyer-side process instance P_b and through messaging, tells CPM *B* to create a seller-side peer process instance P_s.
- *A* dispatches and executes T_1, and upon receipt of the *task return message,* r_1, forwards it to *all* other players of the process, in this case, simply *B*. Both *A* and *B* update their process state and schedule the possible next step of their own peer process instance based on that message.
- When *A* proceeds to activity T_2, since the role represented by *A* does not match the role of

T_2, *A* simply waits for the peer server, that is B in this example, to handle it at the peer site.

- When *B* proceeds to activity T_2, since the role represented by *B* matches that of T_2, T_2 will be handled by *B*.
- The execution of peer process instances at both peer CPMs progress in this way, towards their ends.

4.2 Synchronizing Process Data and Data in the Task Return Messages

An activity is dispatched to a software agent or a human user to perform, and upon its termination, a task return message is sent back and used to trigger the next step in process execution. Such a task return message contains the following information:

- coop-key of the logical process instance,
- handles (local Ids) of the process instance, task, and activity,
- activity execution status,
- the sub-packet, i.e. the subset of process data passed to the activity.

When a task return message comes back to the local CPM engine, it contains all the above information. Since the sub-packet of the process data passed to the activity may be updated during task execution, it must be reconciled with the process data packet after returning. However, before such a message is forwarded to a peer player, only the updated data elements that are shared with the player are retained. (Recall that the sharing scope of each data element is specfied in the process definition.)

4.3 Queuing based Message Delivery Synchronization

A key design issue is to maintain the right order of message processing. For various reasons the messages may not be delivered in the original order. Refer to Figure 9 for the following scenario, for example.

(1) CPM *A* initiated a process instance P_A, and then started executing the first task, T_1;
(2) CPM *A* informed CPM *B* to create and execute the peer process instance, P_B, soon after initiating P_A;
(3) Upon completion of T_1, *A* forwarded the task return message of T_1 to CPM *B*.

A possible consequence caused by out-of-order message delivery is, when the task return message of T_1 reaches CPM *B*, the initiation of P_B has not completed yet, thus there is no ground for processing the above message.

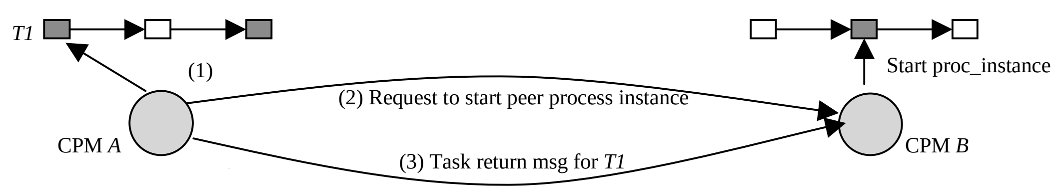

Figure 9: *Task return message received from peer before the counterpart process instance ready*

As another example, consider the execution of a collaborative process with three peer instances run by CPMs *A*, *B* and *C*, responsible for tasks T_1, T_2 and T_3 respectively. These tasks are to be executed in the order T_1, T_2, T_3 . Please refer to Figure 10 for the following scenario.

(1) When task T_1 run by CPM *A* completed, *A* forwarded the task return message of T_1, msg_1, to both *B* and *C*;
(2) Upon receipt of msg_1, CPM *B* started executing task T_2;
(3) When T_2 completed, *B* forwarded the task return message of T_2, msg_2, to both *A* and *C*.

In this scenario, a possible consequence caused by out-of-order message delivery is, when msg_2 reached *C*, it hasn't received msg_1. In this case, processing msg_2 at CPM *C* can lead to an inconsistent result.

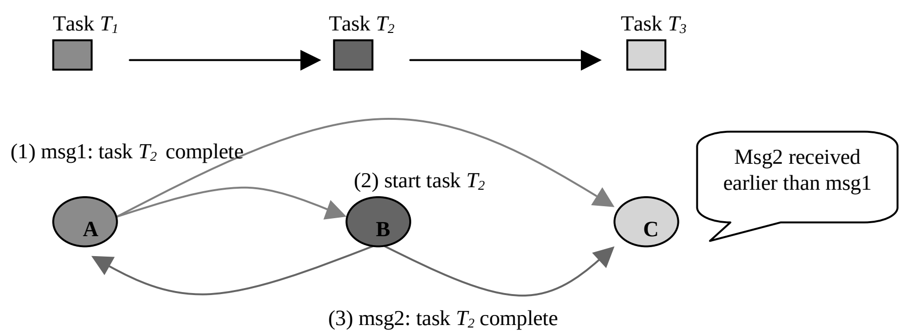

Figure 10: *Task return messages from peer CPMs received in wrong order*

Queuing technique and the knowledge drawn from process definitions are used to resolve the out-of-order message delivery problem. Each CPM has a **queuing server**, in additional to the regular message queue handler (Figure 11). This queuing server is *workflow specific* as it interfaces to the process definition handler and the process instance log handler, using process definitions and execution histories to make operational decisions. It also responds to CPM internal events such as process instance status changes.

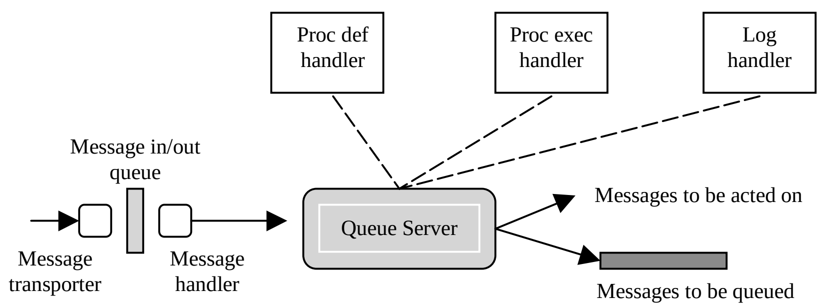

Figure 11: *Queuing server of CPM*

The general functions of the queuing server include the following.

- When a message is received, check if it is ready to be processed based on the process definition, execution history and queued messages, and if not, queue the message. In the above example, if msg_2 for task T_2 cannot be executed at CMP *C* since *C* hasn't received the task return message for task T_1, msg_2 is to be put in the queued first.
- After a new message is processed, check if any queued message is ready to be processed as a result, and if there is, process it. In the above example, assume that CPM *C* queued msg_2 for

task T_2 since it did not receive the task return message, msg_1, for T_1. Later, when msg_1 was eventually received, CPM C would process msg_1 for T_1 first, followed by processing msg_2 for task T_2.

- When a CPM internal event about process instance status change (e.g. started, terminated, suspended) is received, the queuing server check if the change makes any queued message ready to be processed. In the example shown in Figure 8, assume that the task return message for T_1 was queued as a result of the unavailability of P_B, upon receipt of the event on P_B's availability, the queuing server enables the processing of that message.

5. An Agent-based Implementation Architecture for CPM

We have implemented CPM and integrated it into a dynamic software agent platform, *E-Carry*, that we have developed at HP Labs. This novel integration achieves two purposes: on the one hand, it provides an implementation and execution platform for a CPM system; on the other hand, it elevates multi-agent cooperation in E-Carry from the conversation level to the process level for mediating e-Commerce applications. In addition, we have also integrated E-Carry with E-speak, an inter-enterprise communication infrastructure, to provide for inter-domain communication for inter-enterprise business processes. In this section, we briefly describe the integration of CPM, E-Carry, and E-Speak. Section 5.1 describes the integration of CPM and E-Carry. Section 5.2 describes the integration of the agent-embedded CPM with E-Speak. Section 5.3 discusses how this implementation architecture also serves the dual purpose of elevating multi-agent cooperation from the conversation level to the process level.

5.1 Agent Embedded CPM

E-Carry is a dynamic and scalable platform for developing agent-based applications [6]. Every E-Carry agent contains a built-in *Service Tier.* The Service Tier of an E-Carry agent has the ability to load applications dynamically and to communicate with other E-Carry agents in the same domain, i.e., within a domain, E-Carry agents sharing the same *Coordinator*. In addition, the service tier contains an embedded Web server with servelet functionality, enabling the state of an E-Carry agent to be accessed through a browser. The development of E-Carry is motivated by providing a migration from the traditional agent infrastructure to a scalable, dynamic and distributed middleware framework.

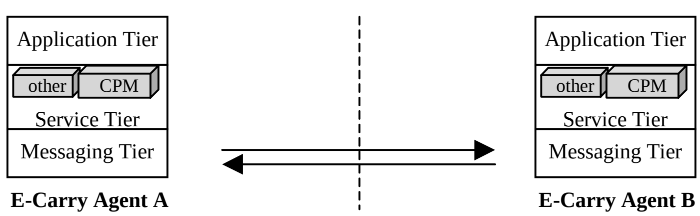

Figure 12: *E-Carry agent with embedded CPM*

We have implemented the functionality of CPM and *embedded* into the service tier of E-Carry, as shown in the following figure. An agent with CPM embedded is used as a CPM server.

However, since a CPM server can also be viewed as an agent, it is possible to consider the notion of *personalized CPM engine*. That is, each logical entity of an enterprise, say, a *complementary product buying agent,* could have its (or his) own CPM engine to represent it (or him) when participating in inter-enterprise collaboration, having its (or his) own CPM server executing peer process instances. Besides of acting as a CPM server, an E-Carry agent can also perform activities. The full details about E-Carry will be reported separately.

Figure 12 shows agents with embedded CPMs. The CPM embedded in an E-Carry agent interacts with the hosting agent though a set of internal messages. The communication between agents is made through inter-agent message exchange. A set of agent messages specific to collaborative process management, are defined, and a corresponding message interpreter is provided for each agent. The E-Carry agent has the capability to load and switch interpreters based on message ontology types thus can easily handle applications in different contexts.

5.2 Inter-Enterprise Agent Communication

Agent in the same group, referred to as the agent **domain**, can communicate using the naming service provided by the *coordinator* of that domain. However, agents in different enterprises may not form a single agent domain. Instead, they need certain "service bus" to locate each other for peer-to-peer communication. Further, issues such as firewall, security, access control, and even billing, should be taken into account. We have adopted the HP E-Speak service bus, an interface based service provisioning and invocation framework with multiple interconnected *E-Speak Cores*. An E-Speak core provides a set of predefined and extensible **infrastructure services** including authentication, authorization, billing, etc. These infrastructure services represent the major difference between E-Speak and the traditional CORBA-like middleware. In this paper we do not intend to explain E-Speak in detail.

Intuitively, any agent, *A*, can register a "send message" service with E-Speak, making it possible for another agent in a foreign domain to directly invoke this service and thus able to directly send a message to *A.* However, if every agent has to register a messaging service in order to receive messages, and every agent has to maintain multiple client side messaging service implementations of all the agents it may need to have a contact with, it is not scalable.

In order to unify the messaging interface for inter-enterprise agent communication, we only register the *messaging service* of the *coordinator* of an agent domain with E-Speak. This service then becomes the single entrance to the agent domain. Inside the domain, the coordinator can forward messages to other agents through intra-domain agent communication. Thus, it is unnecessary for each agent to register an individual message service, and the coordinator provides a gateway for any foreign agent to reach that agent-domain. On the other hand, every E -Carry agent only needs to be provided with a "standard" interface and the client-side stub for invoking the above messaging service, using the agent domain name as a parameter. By invoking the message service of an agent domain, an agent can contact any agent in that domain, with messages routed by the coordinator of that domain.

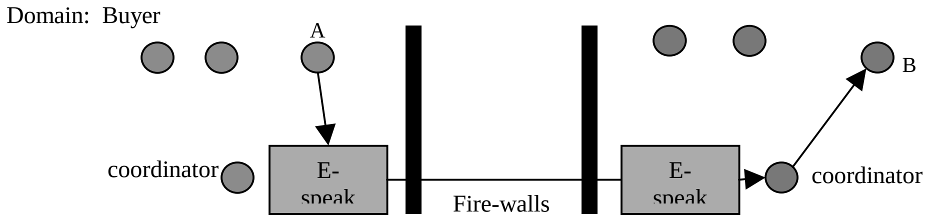

Figure 13:*Unify messaging service interface to simplify inter-enterprise agent communication*

This mechanism is actually transparent at the message level. In an intra-domain message, the destination is simply expressed by the receiver's name. In an inter-domain message, it is expressed by

espeak:domain_name/agent_name

Here *espeak* is used to identify the service bus, a concept at a higher-level than transport service. Given a *domain_name*, the messaging service of the coordinator of that domain can be invoked, as the messaging gateway for contacting all the agents in that domain. The *agent_name* local to the domain is then used by the coordinator to route messages to that agent. Refer to Figure 13, when agent *A* sends a message to agent *B* in domain "*Vendor*", the full address of *B* is *espeak:Vendor/B*, and the message is transferred through E-Speak infrastructure to the coordinator of domain *Vendor*, then forwarded to *B*.

5.3 CPMs for Process-Level Agent Cooperation

The collaboration of multiple peer CPMs is analogous to multi-agent cooperation [1,6,18,20,23,24,29]. In fact, using agent technology to support E-Commerce automation is a promising direction [4,5,22,26,27,30]. However, the previous "proof-of-concept" efforts in agent platforms do not scale well in E-Commerce automation for the following two major reasons.

- Most E-Commerce applications are based on inter-enterprise business partnership, but the current mechanisms for multi-agent cooperation is based on intra-enterprise coordination, without addressing the issue of inter-enterprise collaboration. The conventional group-based coordination cannot handle inter-enterprise agent cooperation, since agents across enterprise boundaries are unlikely to be organized into the same group and under a centralized coordination.
- In the conventional agent platforms, agents cooperate through message exchanges, or *conversation*s [17,18,22]. However, many real applications include complex business processes with a number of concurrent, long-duration, nested tasks, which are difficult to manage and trace through flat conversations. Instead, a more robust and scalable approach is to lift agent cooperation from the conversation level to the process level.

Turning agent cooperation from conversation-level to process-level is a natural and necessary move. In general, businesses collaborate by following certain rules, such as "if you send me a price request then I will send you a quote", and "if the quote I sent you is acceptable, then you will send me an order". These rules include sequences of steps, with some of those steps nested.

Such business collaboration usually involves multiple agents, each responsible for managing or performing certain tasks that contribute to the process. Adding a process-level coordination capability into agent-based systems is critical in enabling the latter to better tackle such applications.

We have relied on the proposed approach to tackle these issues. The combination of E-Carry and CPM allows us to scale agent cooperation from conversation level to process level, and from intra-enterprise cooperation to inter-enterprise collaboration.

6. Conclusions

Focusing on *inter-enterprise* E-Commerce automation at business process level, we have developed the c*ollaborative process manager* (CPM) to support peer-to-peer process management. We further embedded CPM into a dynamic software agent architecture, E-Carry, that we developed at HP Labs, and extended E-Carry with the inter-domain communication capability by utilizing inter-domain messaging services such as E-Speak and by introducing inter-domain messaging protocol. Through this work, we have made conceptual as well as practical contributions to both workflow technology and agent technology.

From the workflow point of view, the proposed approach can be used to enhance the collaboration of business partners, and to support inter-enterprise business processes, a practical extension to the current workflow approach.

From the multi-agent system point of view, the proposed approach can be used to lift agent cooperation from the conversation level to the process level, and from centralized coordination to peer-to-peer collaboration. The combination of CPM and agent framework can be a step towards a scalable, dynamic, inter-enterprise middleware framework.

The feasibility of this approach has been demonstrated in a prototype implemented at HP Labs. We are currently investigating the use of this infrastructure to support CBL (Common Business Library)- and RosettaNet-based E-Commerce automation.

REFERENCES

[1] Aglets, "Programming Mobile Agents in Java", IBM, http://www.trl.ibm.co.jp/aglets/, 1997.

[2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, “Extensible Markup Language (XML) 1.0 Specification”, February 1998, (http://www.w3.org/TR/REC-xml)

[3] A.Buchmann, M.Ozsu, M.Hornick, D.Georgakopoulos, and F.A. Manola"A transaction model for active distributed object systems", A. Elmagarmid (ed) Transaction Models for Advanced Database Applications, Morgan-Kaufmann, 1992.

[4] A. Chavez and P. Maes, Kasbah: An Agent Marketplace for Buying and Selling Goods, Proc. of PAAM96, 1996.

[5] Q. Chen, Meichun Hsu, Umesh Dayal, Martin Griss, “Incorporating Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation”, Proc. Fourth International Conference on Autonomous Agents, 2000, Span.

[6] Q. Chen, P. Chundi, Umesh Dayal, M. Hsu, "Dynamic-Agents", International Journal on Cooperative Information Systems, 1999, USA.

[7] Q. Chen and Umesh Dayal, "Failure Recovery across Transaction Hierarchies", Proc. of 13th International Conference on

Data Engineering (ICDE-97), 1997, UK.

[8] Q. Chen and Umesh Dayal, "A Transactional Nested Process Management System", Proc. of ICDE-96, 1996.

[9] Q. Chen and Umesh Dayal, "Commit Scope Control in Nested Transactions", Proc. of EDBT'96, 1996, France.

[10] Q. Chen, Umesh Dayal, "Contracting Transaction Hierarchies", Proc. of RIDE'96, 1996, USA.

[11] P.K. Chrysanthis and K.Ramamritham"Acta: The saga continues", A. Elmagarmid (ed) Transaction Models for Advanced Database Applications, Morgan-Kaufmann, 1992.

[12] CORBA, "CORBA Facilities Architecture Specification", OMG Doc 97-06-15, 1997.

[13] U. Dayal and M. Hsu and R. Ladin, "Organizing Long Running Activities with Triggers and Transactions", Proc. ACM-SIGMOD'90, 1990.

[14] U. Dayal and M. Hsu and R. Ladin, "A Transactional Model for long Running Activities", Proc. VLDB'91, 1991.

[15] Document Object Model, http://www.w3.org/DOM/

[16] E-Speak, http://www.e-speak.net/

[17] T. Finin, R. Fritzson, D. McKay, R. McEntire, "KQML as an Agent-Communication Language", Proc. CIKM'94, 1994.

[18] Foundation for Intelligent Physical Agents(FIPA)- FIPA97 Agent Specification, http://www.fipa.org/

[19] J.Gray and A.Reuter, "Transaction processing: Concepts and techniques"Morgan Kaufmann Publishers, 1993.

[20] R. S. Gray. Agent Tcl: A flexible and secure mobile-agent system. Dr. Dobbs Journal, 22(3):18-27, 1997.

[21] S.Heiler, SHaradhvala, S.Zdonik, B.Blaustein, and A.Rosenthal"A flexible framework for transaction management in engineering environments", A. Elmagarmid (ed) Transaction Models for Advanced Database Applications, Morgan-Kaufmann, 1992.

[22] N.R. Jennings, P. Faratin, M.J. Johnson, P.O'Brien & M.E Wiegan, "Using Intelligent Agents to Manage Business Processes". Proc. of PAAM96, U.K., 1996, pp. 245-360.

[23] N. R. Jennings (1999) "Agent-based Computing: Promise and Perils" Proc. IJCAI-99, Sweeden. 1429-1436.

[24] T. John, Intelligent Agent Library/Factory, release 4 (http://www.bitpix.com)

[25] N.Krishnakumar and A.Sheth"Specification of workflows with heterogeneous tasks in meteor", Proc. VLDB'94, 1994.

[26] P. Maes, R. H. Guttman and A. G. Moukas, "Agents that Buy and Sell", CACM 42(8), March, 1999.

[27] A.G. Moukas, R. H. Guttman and P. Maes, "Agent Mediated Electronic Commerce: An MIT Media Laboratory Perspective", Proc. of International Conference on Electronic Commerce, 1998.

[28] M.Nodine and S.Zdonik"Cooperative transaction hierarchy: A transaction model to support design applications", Proc. of VLDB'90, 1990.

[29] Odyssy, "Agent Technology: Odyssey", General Magic, http://www.genmagic.com, 1997.

[30] B. Perry, M. Talor, A. Unruh, "Information Aggregation and Agent Interaction Patterns in InfoSleuth", Proc. of CoopIS'99, UK, 1999.

[31] Rosetta-net, www.rosettaNet.org.

[32] M. Rusinkiewicz, W. Klas, T. Tesch, J. Wasch, P. Muth, "Towards a Cooperative Transaction Model – The Cooperative Activity Model", VLDB'95, 1995.

[33] H.Wachter and A.Reuter, "The contract model", A. Elmagarmid (ed) Transaction Models for Advanced Database Applications, Morgan-Kaufmann, 1992.

[34] Workflow Management Coalition, www.aiim.org/wfmc/mainframe.htm.