Peer-to-Peer Collaborative Internet Business Servers

Qiming Chen, Meichun Hsu, Umesh Dayal
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-14
January 18th , 2001*

E-mail: {qchen, mhsu, dayal}@hpl.hp.com

As the Web evolves from providing information access into a platform for e business, there is a need for technologies to support dynamic inter-enterprise collaboration. Toward this goal, we have developed **Internet Business Servers** (IBSs). This approach essentially turns individual Web servers into communicating and collaborating servers, with extended functionality for inter-enterprise business collaboration. An IBS embeds a Web server, and, therefore, is accessible via any Web browser anywhere. IBSs support **Peer-to-Peer** (P2P) computation at multiple tiers. At the communication tier, IBSs can communicate using heterogeneous messaging protocols and transport mechanisms. At the business process tier, IBSs collaborate by means of built-in **Conversational Process Managers** (CPM) that provide decentralized, P2P process management. At the business tier, IBSs support a document based e-commerce model.

IBS represents the merging of multiple disciplines such as Web technology, agent technology and workflow technology. However, it extends agent technology by elevating agent cooperation from the conversation level to the process management level; extends workflow technology from centralized process management to decentralized conversational process management; and extends Web technology by turning individual Web servers into communicating and collaborating servers. Further, integrating the advanced P2P collaboration capabilities we have developed with existing P2P networking and resource sharing technologies into a single autonomous system platform represents a step towards a general and dynamic P2P-ware infrastructure.

* Internal Accession Date Only Approved for External Publication

Peer-to-Peer Collaborative Internet Business Servers

Qiming Chen, Meichun Hsu, Umesh Dayal

HP Labs

Hewlett Packard Co.
1501 Page Mill Road, MS 1U4
Palo Alto, California, CA 94303, USA
+1-650-857-3060

{qchen, mhsu, dayal}@hpl.hp.com

Abstract

*As the Web evolves from providing information access into a platform for e-business, there is a need for technologies to support dynamic inter-enterprise collaboration. Toward this goal, we have developed **Internet Business Servers** (IBSs). This approach essentially turns individual Web servers into communicating and collaborating servers, with extended functionality for inter-enterprise business collaboration. An IBS embeds a Web server, and, therefore, is accessible via any Web browser anywhere. IBSs support **Peer-to-Peer** (P2P) computation at multiple tiers. At the communication tier, IBSs can communicate using heterogeneous messaging protocols and transport mechanisms. At the business process tier, IBSs collaborate by means of built-in **Conversational Process Managers** (CPM) that provide decentralized, P2P process management. At the business tier, IBSs support a document based e-commerce model.*

IBS represents the merging of multiple disciplines such as Web technology, agent technology and workflow technology. However, it extends agent technology by elevating agent cooperation from the conversation level to the process management level; extends workflow technology from centralized process management to decentralized conversational process management; and extends Web technology by turning individual Web servers into communicating and collaborating servers. Further, integrating the advanced P2P collaboration capabilities we have developed with existing P2P networking and resource sharing technologies into a single autonomous system platform represents a step towards a general and dynamic P2P-ware infrastructure.

The feasibility and practical value of this work have been demonstrated through prototypes implemented at HP Labs.

1. Introduction: From Information Super-Highway to Business Super-Highway

E-Business applications operate in a distributed environment involving multiple parties with different capabilities and dynamically changing characteristics such as performance and availability. Very often, a business application uses services and components provided by multiple enterprises, and this business relationship may be created dynamically and maintained only for the required duration such as a single transaction. Creating and managing such dynamic

e-business applications requires an autonomous and decentralized environment that supports inter-enterprise collaboration at several levels [3,4,6-8,24,25]. First, it must support the discovery and provisioning of services, and communication among services, across enterprise boundaries. Second, it must support collaboration among business processes in different enterprises. Third, it must support business information exchange and sharing among enterprises.

Consider, for example, an electronic purchasing application. This involves multiple enterprises: a buyer and one or more vendors. There must be some mechanism for the vendors to advertise and register their services (in this case, their product catalogs, for example), for the buyer to discover the appropriate vendors, and for the various parties to communicate with one another.
At the business process level, coordination is required among activities that are performed by the different parties: for example, the buyer sends out a request for a quote; the vendors respond with quotes; the buyer selects one or more vendors, generates and sends out purchase orders; the vendors process the purchase orders, dispatch the products ordered, and send invoices; the buyer receives the products and processes the invoices; and so on. Each of these activities may itself involve complex business processes within each participating enterprise, and these must be coordinated to accomplish the entire collaborative purchasing process. At the business level, the parties collaborate via exchange of documents (e.g., request for quote, invoice), which must conform to agreed upon business concepts and formats.

Today, Web servers provide an autonomous and decentralized environment, but their primary function is content delivery (information access). Although the functionality of a Web server can be enhanced using active pages (servlets) and active proxies, there is little support for service composition and business collaboration. With the goal of evolving the Web to support e-business applications, we have developed the concept of **Internet Business Servers** (IBSs), which allows us to extend the functionality of Web servers beyond content delivery to inter-enterprise collaboration.

The design of IBS preserves the general principles of Web server design, such as being autonomous and decentralized, supporting message-based communication, and handling inter-enterprise message delivery. Also, since an IBS embeds a Web server (or alternatively, is linked to a Web server as its servlet), it is accessible via any Web browser from anywhere. Focusing on mediating and integrating loosely coupled e-business components and applications, IBSs are more active than Web servers, and are capable of monitoring the environment and initiating actions and processes on behalf of their “owners”, which might range from business applications to service providers to individual users.

IBSs support **peer-to-peer** (P2P) computation, a new direction of distributed computation [18,27]. The current focus of P2P is on networking and resource sharing (e.g., music swapping tools or PCs), without any centralized server, with the goal of enhanced reliability, security, and availability (because a single point of failure won’t take down the infrastructure) and scalability (sharing CPU cycles on idle PCs). Indeed, innovative P2P tools – messaging, search, discovery, document exchange – have value in and of themselves. However, in order to leverage them to

build scalable e-business solutions, a general-purpose platform providing a consistent set of system level services, is required. Therefore, we focus on providing integrated system facilities in the form of an *autonomous but extensible platform* for dynamic application support, service provisioning, information transfer and synchronization, and collaborative task coordination and process management (e.g., workflow) that can be applied consistently across a spectrum of tools and applications.

IBSs support P2P collaboration at multiple tiers: communication tier, process collaboration tier, and business collaboration tier. At the communication tier, IBSs can communicate using heterogeneous **messaging protocols** and **transport mechanisms**. IBSs collaborate at the process tier by means of a built-in **Conversational Process Manager** (CPM), which provides decentralized, P2P process management [5]. A conversational process typically involves two or more participants. Very often, these participants represent different enterprises and do not share private data or details of internal processes with each other. A conversational process is based on a common agreement between these parties, and is not executed by a centralized workflow engine. Instead, the execution of a conversational process includes separate peer-process instances handled by peer-CPMs, each representing a participating party in the collaborative business process. While a CPM only schedules, dispatches, and controls those process tasks for which it is responsible, it must synchronize its peer-execution with other CPMs through an inter-CPM messaging protocol. In general, this approach represents a shift from centralized process management to decentralized, P2P conversational process management.

At the business tier, IBSs support a **document services architecture** for e-business, in which "shared document definitions provide a framework for specifying the business logic and computations that take place on each end of a document exchange" [11]. IBSs are provided with the Common Business Library (CBL) that defines business concepts and document templates, and enables business collaboration through document exchange, which is incorporated with the execution of *conversational processes*. This allows us to elevate integration from the system level to the *business level*, with interfaces that remain stable, despite changes in internal implementation, organization and process.

Given the above capabilities, IBSs can be used to support a wide array of new applications in collaborative and dynamic resource allocation, service discovery and utilization. For example, some properties of services, such as current availability, system load and idle capacity, cannot be pre-registered and advertised, but can be discovered on the fly through P2P communication and collaboration. Service utilization is not just a matter of a client invoking the service, but involves P2P collaboration through multi-step conversations and process coordination between the provider of the service and the user of the service. When businesses are built on one another's services, the boundary between server and client becomes blurred.

IBS represents the merging of multiple disciplines. Besides extending Web technology as discussed above, IBS also significantly extends agent technology [1,9,15-17,20,22,26,28] and workflow technology [19,21,23,34]. Existing "proof-of-concept" agent systems do not scale well

for e-business automation. An essential reason is that the conventional agent infrastructures are primarily designed for intra-enterprise, group-based agent cooperation, whereas most e-business applications are based on *inter-enterprise* business partnership. Agents across enterprise boundaries are not likely to be organized into the same "agent group" and under centralized coordination. Similarly, conventional workflow systems are primarily designed for *intra-enterprise* process management, and they are inadequate for handling processes with tasks and data separated by enterprise boundaries, for reasons such as security, privacy, data sharing, firewalls, etc. In fact, the cooperation of multiple enterprises is often based on P2P interactions rather than on centralized coordination. As a result, the conventional centralized process management architecture does not fit the picture of inter-enterprise business-to-business applications.

We envisage the development of IBSs as a way to turn the **information super-highway** into a **business super-highway**. The information super-highway is connected by autonomous and decentralized Web servers. Analogously, the business super-highway is connected by autonomous, decentralized and also *collaborative* IBSs. The feasibility and practical value of this work have been demonstrated through prototypes implemented at HP Labs.

The features of IBSs will be described tier by tier. Section 2 illustrates the general architecture of IBS. Sections 3, 4 and 5 cover the tiers for P2P communication, P2P process collaboration and P2P business collaboration respectively. Section 6 illustrates the use of IBSs for collaborative dynamic service discovery. Finally, some concluding remarks are given in Section 7.

2. Architecture of an Internet Business Server

IBSs include the features of Web servers, but extend them with service provisioning, composition and collaboration capabilities. To support general e-business applications, it is neither sufficient for the IBS to be equipped with a fixed set of application-specific functions, nor practical for the above capabilities to be developed from scratch for each IBS. This has motivated us to develop IBS as a general-purpose infrastructure, rather than as an application-specific system. The IBS is a *dynamic platform* with the capability of carrying different services dynamically and modifying its behavior on the fly. Thus, for example, an IBS can flexibly mediate either a buyer or a seller by loading, carrying and running different programs. In cooperative work, such dynamic role assignment and switching allow an IBS to play different roles while maintaining its identity and consistent communication channels, as well as retaining data, knowledge and other system resources to be used across applications. (In this respect, IBSs are quite similar to the dynamic agent infrastructure we introduced in [9].) These features represent the fundamental difference between IBSs and traditional agents, as the latter normally have their abilities fixed at the time of initiation. These features also differentiate IBSs from regular Web servers in facilitating a business layer.

As an autonomous system, an IBS provides a general execution environment for a set of built-in

servers, which offer the ability of loading and running applications dynamically, storing objects, communicating with other IBSs based on domain specific ontology and switchable message interpreters, and accomplishing collaboration at the business process and document exchange levels.

As shown in Figure 1, every IBS contains an embedded Web server with servlet functionality, which allows the state of an IBS to be accessed through a browser, and allows the IBS to send and receive messages using HTTP. Every IBS also has an embedded Conversational Process Manager (CPM) that underlies the P2P collaboration at the business process level across enterprise boundaries [5].

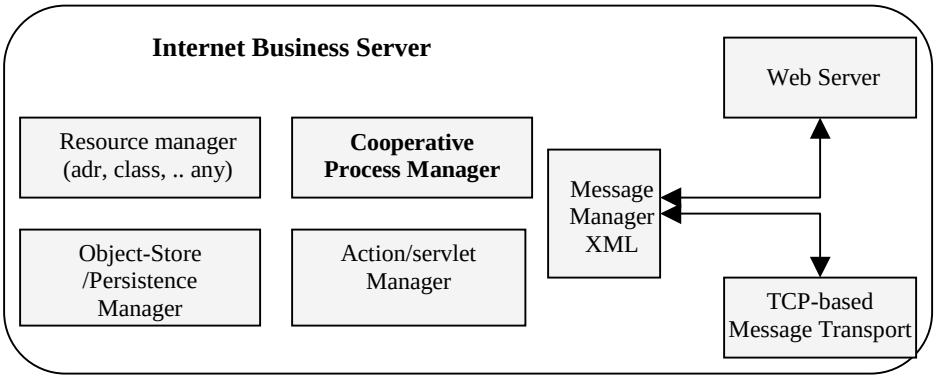

Figure 1: *Architecture of IBS*

The dynamic behavior of an IBS is supported by two kinds of servlets: the servlets of the embedded Web server, and the servlets of the IBS. The dynamic extensibility of IBS is similar to Web servlet capability but with the following additional strengths.

- Unlike Web servlets, IBS servlets, or *carried applications*, can be any Java programs that the IBS downloads, instantiates and executes; this overcomes the limitation of Web servlets, which are only accessible through a Web server. A carried application, when started, is offered a reference to the underlying built-in management facilities (such as persistent object management, resource management, communication), and can use this reference to access the APIs of the facilities.
- The object storage capability of IBS can easily provide state control for Web servlets. As we know, one feature of Web servlets is the re-entry of dynamically formed HTML pages with hidden variables as input arguments. Since servlets are largely "stateless", these hidden variables are used to store the "state" information needed by the servlet. Such an approach may be error-prone without proper control based on object states maintained outside the servlets. As shown in Figure 2, the work-item objects (representing tasks) generated by the IBS-embedded CPM can be maintained in the embedded object-store, and operated by the servlets supported by the embedded Web server, while the task states are kept in the object-

store for providing the necessary state-control for the servlets.

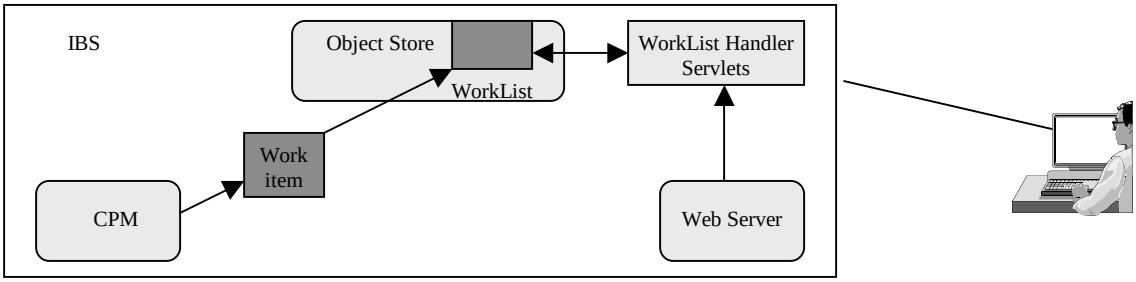

Figure 2: Cooperation of *IBS's built-in components*

In addition, in contrast toWeb servers, IBSs in the same enterprise can form groups, which we call *IBS domains*. IBSs in the same domain can communicate using the naming service provided by the *coordinator* (which is itself an IBS) of that domain. Besides a naming service, other coordination services such as resource brokering, event brokering and request brokering, if required, may be provided either by the coordinator or by other designated IBSs. Further, IBSs may form hierarchical groups within a domain, with a coordinator for each group. Based on the group hierarchy a multilevel name service can be built.

The essential characteristic of IBS is the capability of **P2P collaboration at multiple tiers**. As shown in Figure 3, IBSs collaborate at three tiers: the communication tier, the process management tier, and the business object tier.

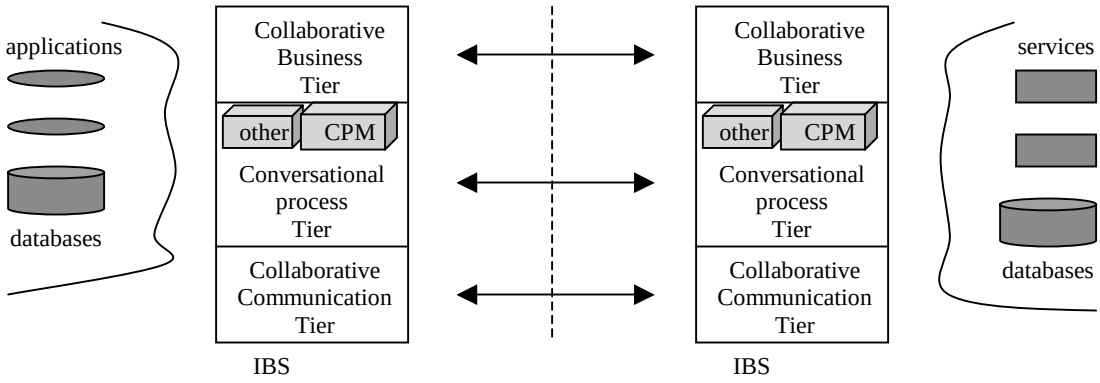

Figure 3: *IBSs collaborating at three tiers*

- At the communication tier, P2P communication is facilitated by the properties of service-bus-independence and message-protocol-independence. This allows IBSs in different domains (or enterprises), which may use different service-buses and different message protocols, to communicate. One IBS can send messages to another using any service-bus (e.g., HTTP, E-Speak [14]), regardless of the default service-bus setting of the receiver. Further, messages self-describe the protocol for interpreting their contents. By loading different interpreters (in the form of Java classes), an IBS can interpret SOAP (Simple Object Access Protocol)

[31,32] messages, regular HTTP messages, XML messages, etc., and with different specialized XML message interpreters, it can further interpret XML messages for specific applications such as on-line shopping, music exchange, etc.

- At the process tier, the embedded CPMs handle business processes collaboratively, thus elevating inter-enterprise cooperation from the conversation level (i.e., simple exchange of messages) to the business process level. Thus, IBSs move beyond centralized process management to peer-to-peer cooperative process management.
- At the business object tier, IBSs support Document Object Model (DOM) and document level business integration. An IBS maintains XML-based Common Business Library (CBL) as Java classes, and exchanges common business objects with other IBSs through regular messages or business process data sharing. Thus, IBSs elevate business integration from the system level to the *business level*, via interfaces that remain stable despite changes in internal implementation, organization and process.

3. P2P Communication Tier

A key requirement for e-business is inter-enterprise communication. Because enterprises are autonomous, inter-enterprise communication has to deal with protocol heterogeneity at both the message transport level and the message semantics level. IBSs support inter-enterprise communication using multiple service-buses and protocols, and support the integration of message-based and interface-based approaches for locating peer IBSs.

Multiple Messaging Service Buses

Within a single enterprise, IBSs can form groups, or *domains*. IBSs in the same domain can communicate using the naming service provided by the *coordinator* (which is itself an IBS) of that domain.

For reasons of autonomy and scalability, IBSs in different enterprises typically will not form a single IBS domain. The full name of an IBS thus consists of a domain name, which should be globally unique, and a local name, which is unique within a domain. It is not practical to provide an inter-enterprise common coordinator for all domains to offer an inter-domain naming service to locate one another. Instead, we need a "service bus", a concept at a higher level than a transport service, to locate each other for peer-to-peer communication. Further, other requirements for inter-enterprise collaboration such as firewalls, security, access control, and even billing, should be taken into account.

We support multiple service buses. In particular, we support HP E-Speak [14] and HTTP.

Using E-Speak as Service Bus

The HP E-Speak service bus is an interface-based service provisioning and invocation framework with multiple interconnected *E-Speak Cores* [14]. 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 traditional CORBA-like middleware.

Intuitively, any IBS, *A*, can register a "send message" service with E-Speak, making it possible for another IBS in a foreign domain to directly invoke this service and thus be able to directly send a message to *A.* However, with IBS's intra-domain name service, we enable an intra-domain IBS to assume a symbolic name separable from its host name. To preserve this level of indirection, it is not desirable to register every intra-domain IBS's "send message" service at E-Speak, since this will force every IBS to permanently bind to a host name.

To unify the messaging interface of all IBSs in a domain, we only register the *messaging service* of the *coordinator* of a domain with E-Speak. This service then becomes the single entrance to the domain. Inside the domain, the coordinator can forward messages to other IBSs through intra-domain communication. Thus, it is unnecessary for each IBS to register an individual message service, and the coordinator provides a gateway for any foreign IBS to reach that domain. On the other hand, every IBS only needs to be provided with a "standard" interface and the client-side stub for invoking the above messaging service, using the domain name as a parameter. By invoking the message service of a domain, an IBS can contact any IBS in that domain, with messages routed by the coordinator of that domain. This is shown in Figure 4.

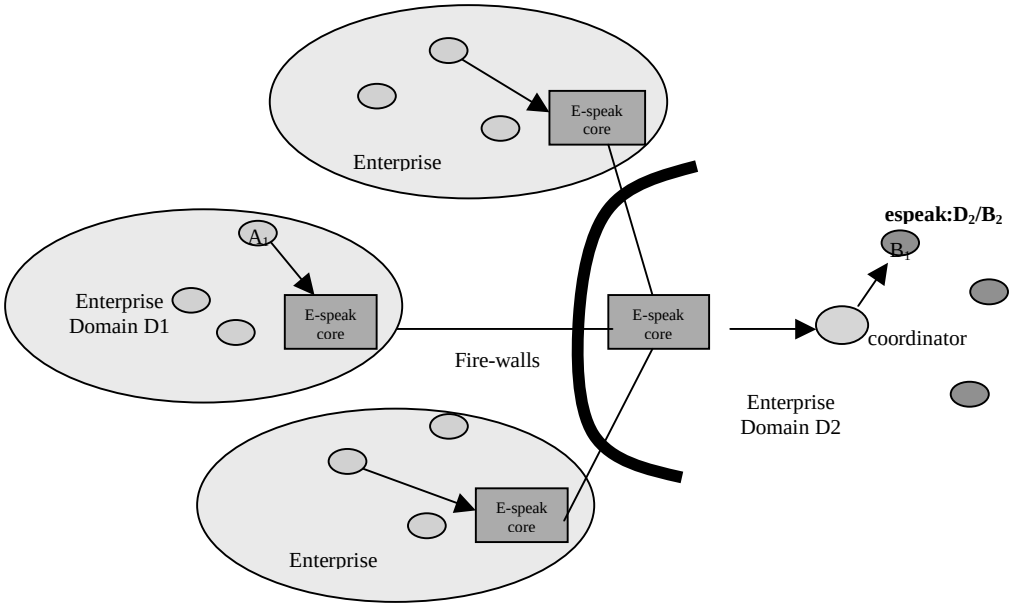

Figure 4: *Inter-domain communication using E-Speak service bus*

Using HTTP as Service Bus

IBSs can also carry out intra-enterprise and inter-enterprise communication using the HTTP POST protocol. The address of an IBS describes two sockets, one for the general TCP transport and the other for the embedded Web server. As shown in Figure 5, when a posted message is received by the Web server, it is treated as the input of a servlet that acts as the message router,

and then the servlet places the message to the message manager of the hosting IBS for processing.

A message sent to an IBS in a foreign domain, usually in a foreign enterprise, is first delivered to the Coordinator, say *C*, of that domain, and then forwarded to the destination IBS.

Locating the Coordinator's URL

There are two ways for the sender to discover the URL of the Coordinator of a domain. If E-Speak is used to register the Coordinator, then one can look up the E-Speak registry. If E-Speak is not used, then an alternative registry, such as one supporting UDDI, or some other community-specific neighborhood protocol, must be used.

To summarize, IBSs can communicate using multiple service buses. In an intra-domain message, the destination is simply expressed by the receiver's name. In an inter-domain message, the destination is expressed by

espeak:*domain_name/IBS_local_name*

or

http:*domain_name/IBS_local_name*

etc. Here *espeak*, or *http*, is used to identify the service bus.

Communication between IBSs is *service-bus independent*. For example, IBS *A* can send a message to IBS B where the default service-bus setting for *A* is E-Speak but for *B* is HTTP. In fact, each IBS domain has a preset, but changeable, service-bus setting as a property of that domain. However, this setting only controls how IBSs *send* messages, but not how they *receive* messages. For receiving messages, each IBS is provided with two ports, one for HTTP post messages and another for general messages through TCP transport. Messages received by any port will be delivered to the *same* IBS message manager for interpretation.

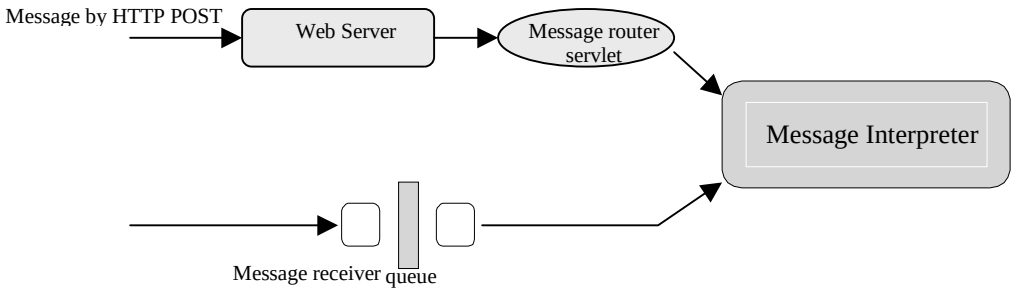

Figure 5: *Support multiple messaging service buses*

Multiple Messaging Protocols

So far, we described support for multiple message transport protocols. However, message

transport protocols do not specify how to parse or interpret the semantics (viz., the content or *payload*) of messages. There exist a number of semantic level protocols such as SOAP [31], OBI, etc. XML is rapidly becoming the standard for specifying the structure of information exchanged between parties [2]. Although XML with Document Type Descriptors (DTDs) provide unified tools for the specification, exchange and usage of information, different application domains have different ontology models, and use different languages and language interpreters, even though the information is in XML format. It is important for a general e-business platform to support multiple such semantic protocols.

Our IBS platform supports *protocol independent* communication. This is because IBS has the capability to load and switch message interpreters based on message ontology types. To support multiple protocols, an IBS can load and carry multiple sets of interpreters corresponding to those protocols. Interpreter invocation and switching are message-driven and made dynamically on the fly. For instance, if the IBS receives a message that is CPM (workflow) specific, it will be so marked in the message envelope, and upon receipt of that message, the CPM interpreter will be invoked automatically to process it. In this way, IBSs can be extended dynamically (by loading additional interpreters) to support other protocols such as SOAP. This capability allows IBSs to easily handle applications in different contexts (such as on-line shopping, electronic purchasing, or collaborative design and manufacturing).

When an IBS participates in multiple applications, it may communicate with other IBSs to accomplish some collaborative task in one application domain, say A_1, using A_1's language and language interpreter; for some other task in another application domain, say A_2, using A_2's language and language interpreter; and so on. In this way, an IBS switches application domains by switching the interpreters it uses.

More intelligently, when an IBS receives a message for processing which it needs a specific interpreter that it does not have, it will ask the sender to send the interpreter or the URL of the interpreter.

4. P2P Process Collaboration Tier

As E-business applications operate in a distributed environment involving multiple parties, they are typically carried out through business-to-business (B2B) and business-to-consumer (B2C) interactions at the business process level. The automation of these activities represents both challenges and opportunities for IBSs in supporting *inter-enterprise P2P business process management*. To support IBS collaboration at the business process level, we have ***embedded*** a P2P ***Conversational Process Manager*** (CPM) into the built-in services of IBS.

To explain the concept of P2P conversational process management, let us first review conventional process management. A business process specifies the integration and synchronization of multiple tasks, or steps; each step represents a logical piece of work, or action, that contributes to the accomplishment of the whole process. Typically a business process

includes a *data packet* containing the *process data* for control flow and data flow, and a task may be given a sub-packet to manipulate. The general function of a workflow engine is to support the definition and execution of business processes. Although the tasks that contribute to a process can be distributed, they are centrally scheduled at the process level. While such centralized process control may be appropriate for a single enterprise, it is not feasible for *inter-enterprise* process management [29]. When multiple parties belonging to different enterprises are involved in a business process, they are unlikely to rely on 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 impediment to using conventional centralized workflow systems for inter-enterprise E-Commerce automation.

In contrast, the execution of a collaborative business process has multiple peer-instances controlled by multiple peer CPMs, and synchronized by the P2P interoperation of the participating CPMs. Accordingly, a CPM is different from the conventional workflow engine in that it only handles the peer execution of a process, and must collaborate with other CPMs for completing the whole process execution. We detailed the notion of conversational process management in [5]. Here, we focus on the integration of CPM with IBS.

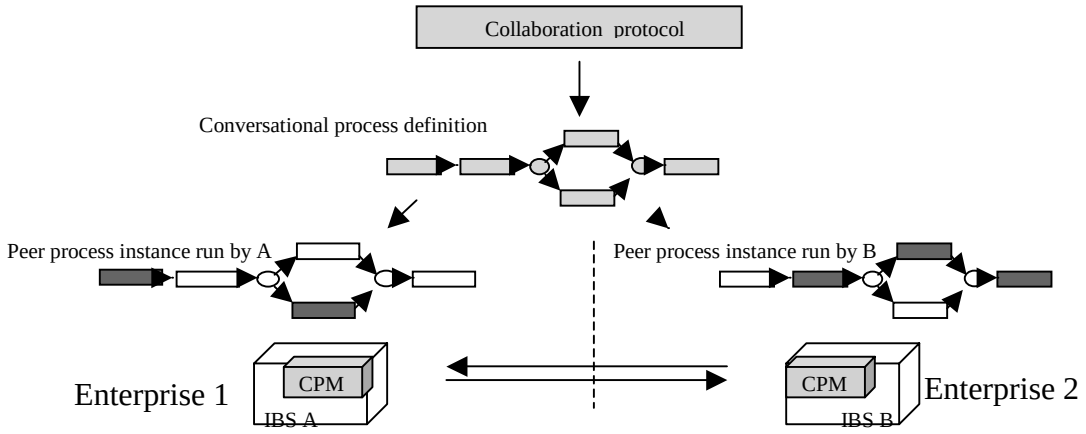

Figure 6: *Peer-to-peer conversational process management*

A conversational process involves multiple parties, and explicitly clarifies the role of each party (Figure 6). The process is defined based on a commonly agreed upon operational protocol, such as the protocol for on-line purchase or auction, and executed by multiple engines collaboratively. More exactly, each execution of a conversational process, or a *logical process instance*, consists of a set of *peer process instances* run by the Conversational Process Managers (CPMs) of participating parties. These peer instances observe 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 that party is responsible for, and the CPMs interoperate through an inter-CPM messaging protocol to synchronize their process execution. An XML based conversational process

Definition Language, XCPDL, extending the process definition language (PDL) [34], is developed for specifying collaborative business processes.

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 to be autonomous, but they comply with the mutually agreed upon purchasing protocol. When *A* finishes a task, it informs *B* of the task status to allow*B* to proceed, and vice versa. The entire purchase process is not handled by any common server, but by the P2P cooperation of multiple servers.

Based on theWFC (Workflow Coalition) standard description [34], a *process* is modeled as a directed acyclic graph (DAG) with nodes representing the steps, or tasks, and arcs representing the flow among those steps. A *work-node* represents a step (*task*) and is associated with an activity, which 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 (a sub-packet) of the process data is passed to it. When it is completed, the sub-packet, possibly updated during the task execution, together with task status information, 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 conversational process, the logical instance of each execution includes multiple peer process instances. Further, a conversational process may have multiple concurrent executions.

To support conversational processes, at the minimum the following must be specified.

- A conversational process has a list of **process-roles**, indicating the logical participants. For example, a simple purchase process with two roles, "buyer" and "seller" is specified as

  ```
  <PROCESS name="OnlineShopping" ...>
    <ROLES> Seller, Buyer </ROLES>
    …
  ```

  Then there will be 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", as shown below.

  ```
  ...
  <WORK_NODE name="T1">
          <ROLE> Buyer </ROLE>
          <DESC> Make PurchaseOrder </DESC>
          <ACTIVITY> PurchaseOrderAction </ACTIVITY>
  </WORK_NODE>

  <WORK_NODE name="T2">
          <ROLE> Seller </ROLE>
          <DESC> Propcess PurchaseOrder </DESC>
          <ACTIVITY> PurchaseOrderResponseAction </ACTIVITY>
  </WORK_NODE>
  ```

If the role of a task is “buyer”, it is only executed in the peer process instance with process-role “buyer”. (Note that these semantics are different from those of task-role and worker-role found in regular business process specifications).

- In an inter-enterprise conversational process execution, each party wants to keep some of the process data private. 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 with the **sharing scopes** of data objects. A template, or a data object specified in 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 conversational 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 with the sharing scopes marked, as shown below.

```
<PROCESS_DATA>
        <ROLE> Seller </ROLE>
        <TEMPLATE> seller_tpl</TEMPLATE>
</PROCESS_DATA>

<PROCESS_DATA>
        <ROLE> Buyer </ROLE>
        <TEMPLATE> buyer_tpl</TEMPLATE>
</PROCESS_DATA>
```

A data packet can be updated or expanded at peer process run time.

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. Furthermore, the process data of the internal sub-process is entirely private to the party executing the sub-process.

A conversational process specification in XML format is first translated into a DOM (Document Object Model) [12,13] tree of Java objects, then into a Java class for cooperative process definition.

An execution of a conversational process consists of a set of peer process instances run by the CPMs, or *players*, 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. 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 launching 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.

In order to maintain the right order of message processing, each CPM has a **queuing server**, in addition to the regular message queue handler. 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.

Turning business 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", in a legal sequence of execution. Such business collaboration usually involves multiple parties, each responsible for managing or performing certain tasks that contribute to the process. Adding a process-level coordination capability into IBSs enables us to support and trace such applications across enterprise boundaries. This approach has also provided a significant extension to the current workflow technology.

5. P2P Business Collaboration Tier

The IBS's capability of carrying data and program objects allows it to maintain and manipulate business objects. As a result, IBSs can support business applications by exchanging business documents, based on the **document service model**.

The general idea of the document service model for e-commerce can be described by the following.

- The types of documents exchanged among trading partners define the *trading community*.
- A *market maker* (such as a trading hub or marketplace) defines the *community standard*.
- The buyers, suppliers, payment acquirers and other service providers can participate in the market if they *produce and consume those documents.*

One example of such a community standard is the Common Business Library (CBL), defined by the Commerce-Net community [10,11].

The document service model elevates inter-enterprise collaboration from the system level to the *business level*, with an interface that remains stable despite changes in internal implementation, organization and process. In fact, defining *business interfaces* in terms of documents rather than APIs allows for an incremental path to business automation: initially, applications can be manually performed through a browser, before being migrated to automated processing.

IBS Collaboration based on Document Service Model

Common Business Library (CBL) is made up of a set of XML building blocks and document

templates commonly used in businesses [10, 11]. It provides the vocabulary (syntax and semantics) for

- *business description primitives*: company, service, product, etc.
- *business forms*: catalog, purchase order, invoice, etc.; and
- *standard measurements*: data, time, location, ...etc.

CBL also provides standard document templates for information interchange among participants. For example, the xCBL 2.0 business document templates published by Commerce-Net includes document templates such as Purchase order, Purchase order response, Order status request, Order status result, Availability check request, Availability check result, Price check request, Price check result, Invoice, Product catalog, Pricing catalog, and so on.

Based on the Document Object Model (DOM), XML-based CBL documents can be easily transformed into Java classes. We have provided a set of Java classes as the counterpart of CBL document templates, and IBSs support CBL-based applications by loading and maintaining that set of classes, together with a set of CBL document processing program classes.

Document-level business rules can be defined based on CBL document exchange. For example, the following two rules are defined based on the exchanges of five types of CBL documents.

- If you send me a *request* for a catalog, then I will send you a *catalog*.
- If you send me a *purchase order* and I can fulfill it, then I will send you an *invoice* and a *shipping notice*.

As we pointed out in the last subsection, besides CBL, IBS can also support other kinds of documents and messages such as SOAP messages.

Collaborative Business Processes for CBL Document Exchange

In the rule examples given above, each rule describes one round of conversation. The “cascaded” firing of such rules actually constitutes a sequence of tasks similar to a simple business process. However, managing P2P applications at the level of individual rules, or single round conversations, does not scale, compared with managing them at the process level, for the following reasons.

- It is usually necessary to maintain and trace the operational status of an application through several rounds of conversation, and to enforce any applicable operational constrains. For example, sending a receipt in response to a payment must take place only after sending an invoice in response to a purchase order; a round of document exchange should not be repeated without a business reason.
- It is necessary to handle concurrent, long-duration, nested tasks for document generation and processing, which are difficult to manage and keep track of as individual rules or single-round conversations.
- There exist complex applications that require certain manual or automated actions in addition to exchanging documents. Such applications cannot be modeled by conversations alone; the

business logic must also be made explicit.

To meet the above requirements, we handle the document exchanges with sequence constraints as P2P conversational processes, in the following way.

- The document generation and exchange steps, e.g. preparing business documents according to appropriate business logic, are handled as process tasks.
- The data container for document exchange is the process data packet. Each peer process instance maintains an individual process data packet. Each task is given a sub-packet, i.e. the subset of process data (e.g. documents) passed to, or generated by, the task.
- The messages for document exchanges are combined with the task return messages.

An activity for a task 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 not only the identification and status information, but also the sub-packet, i.e. the subset of process data passed to, or generated by, the activity.

When a task return message comes back to the local CPM engine, the sub-packet of the process data passed to the activity must be reconciled with the process data packet after returning. However, before such a message is forwarded to a peer player for synchronizing peer process execution, only the updated data elements that are shared with the player are retained (recall that the sharing scope of each data element is specified in the process definition). A forwarded task return message can carry the appropriate business documents to be delivered. This allows us to integrate document exchange with P2P process management.

Support Transactional Business Processes

An important advantage gained from dealing with P2P business interaction (e.g., conversations) at the conversational process level is the ability to preserve transactional properties [30,33]. This can be viewed from the following two aspects.

- For each player, the documents exchanged, including those sent to and received from the peer players, are kept in the process data packet, until all the process instances finish, and then made persistent (upon commit) or dropped (upon abort).
- Upon termination of the process execution, the CPM initiating the process instance, can act as the coordinator for executing the 2PC protocol for synchronizing peer commitment. Introducing such a coordinator ensures the consistency of the commit/abort decision, without assuming the use of any global database.

Note that across enterprise boundaries, only conversational process management can support such transactional process execution. It may not be feasible under centralized process management, since different enterprises may not be able to share a common process data packet.

6. An E-Business Application Example: Collaborative Dynamic Service Discovery

Essentially, IBSs differ from Web servers in their support for collaboration. While Web servers are individual systems, IBSs are communicating and collaborating systems. By contacting a Web server, a user can position the resources under the control of that Web server. By contacting an IBS, a user can request the IBS to discover the resources or services that are maintained, identifiable, or discoverable by other communicating IBSs.

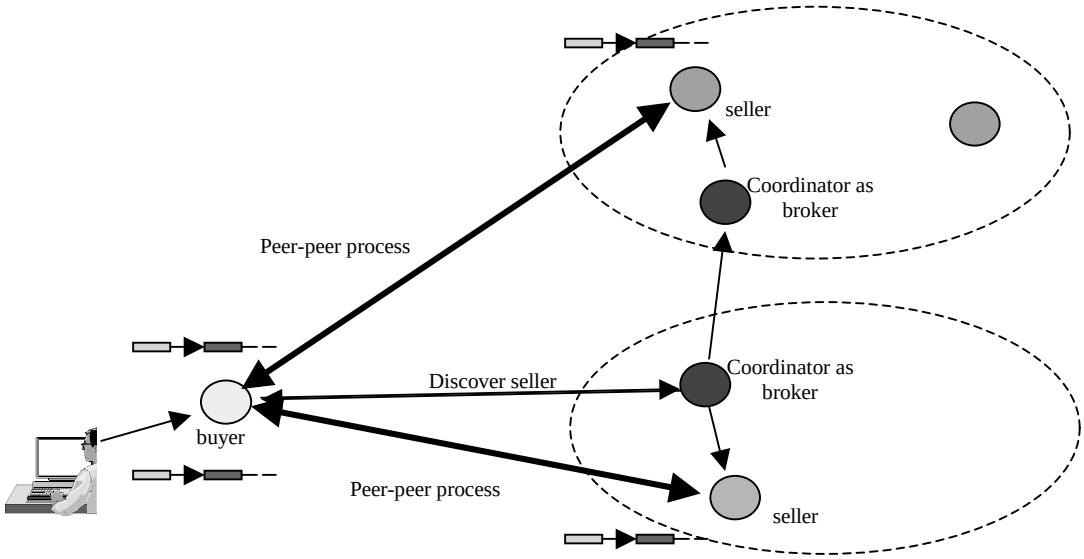

Figure 7: *Dynamic service discovery*

In Figure 7, we show an example of **dynamic service discovery** that we implemented on our prototyping IBS infrastructure**,** in which a service is discovered based on some dynamic properties, such as inventory availability, which cannot be pre-registered and advertised.

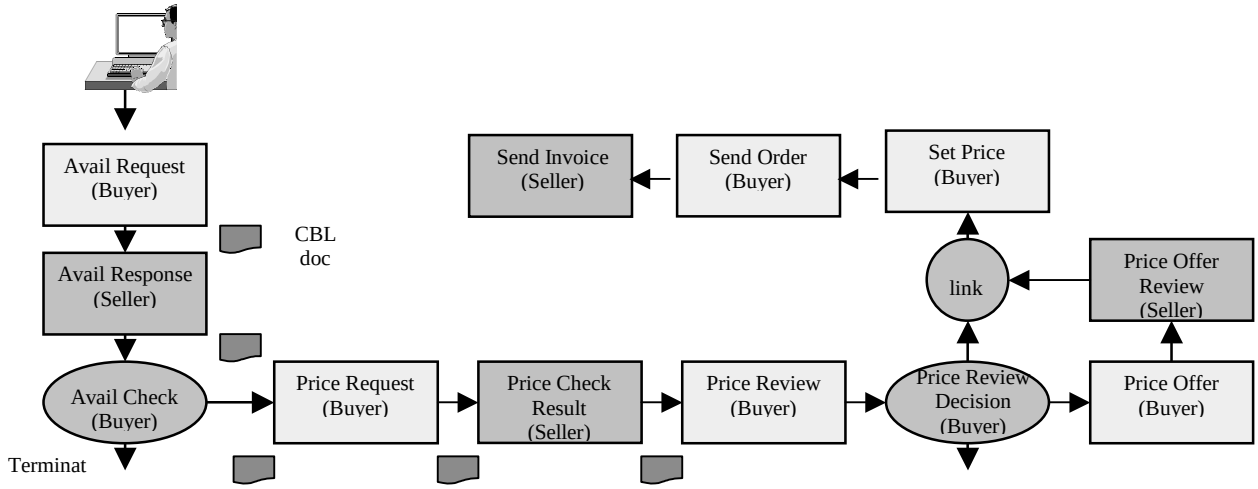

Figure 8: *Peer-to-peer conversational process forpurchasing*

The scenario is as follows:

- A user instructs the IBS playing the role of *buyer* to purchase a product.
- The *buyer* IBS contacts the IBSs playing the role of *broker* in multiple neighborhood domains to find the appropriate *sellers*. If necessary, those *broker* IBSs may further contact their own neighborhood *broker* IBSs.
- These *broker* IBSs discover and select (based on certain filtering criteria) the *seller* IBSs, and send messages to the *buyer* IBS to initiate purchase processes with those seller IBSs.

Multiple conversational process instances are set up between the *buyer* IBS and each *seller* IBS. Figure 8 illustrates the conversational process executed between the *buyer* IBS and each discovered *seller* IBS. Information exchange is carried out through the exchange of CBL documents, and these documents are carried by the task return message received by the CPM and broadcast to peer CPMs for synchronizing peer process executions. When *buyer* IBS compares all the offers to pick up the best offer, it may contact human users through the Web and email for validation and finalization of the deals.

7. Conclusions

With the goal of evolving the Web to support e-business collaboration, we have developed the concept of Internet Business Servers (IBSs), which allows us to turn individual Web servers into communicating and collaborating servers. We extend the functionality of a Web server from content delivery to business collaboration. While Web servers are individual systems, IBSs are communicating and collaborating systems. Further, the IBS infrastructure reflects the P2P computing paradigm. In this paper we have described IBS P2P collaboration at the communication, process management, and business document layers.

The collaboration of multiple peer IBSs is analogous to multi-agent cooperation, but represents a more robust and scalable approach. It can lift agent cooperation from the conversation level to the process level, for dealing with complex business applications with concurrent, long-duration, long-waiting and nested tasks as (optionally transactional) business processes.

Integrating the advanced P2P capability we have described in this paper with existing P2P techniques into a single autonomous system platform, offers IBS the power for scaling Internet-based P2P computing, and represents a step towards a general and dynamic P2P-ware infrastructure. This allows us to enhance the interaction of dynamically formed business partnerships, and to conduct inter-enterprise business collaboration at the process level. The feasibility of this approach has been demonstrated in a prototype implemented at HP Labs. We are currently investigating a more general P2P computing paradigm for e-services.

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

[4] Qiming Chen, Meichun Hsu, Igor Kleyner, "How Agents from Different E-Commerce Enterprises Cooperate", Proc. of The

Fifth International Symposium on Autonomous Decentralized Systems (ISADS'2001), 2001, USA.

[5] Qiming Chen, Meichun Hsu, "Inter-Enterprise Collaborative Business Process Management", Proc. of 17th International Conference on Data Engineering (ICDE-2001), 2001, Germany.

[6] Qiming Chen, Umeshwar Dayal, "Multi-Agent Cooperative Transactions for E-Commerce", Proc. Fifth IFCIS Conference on Cooperative Information Systems (CoopIS'2000), 2000, Israel.

[7] Qiming Chen, Umesh Dayal, Meichun Hsu, Martin Griss, "Dynamic Agent, Workflow and XML for E-Commerce Automation", Proc. First International Conference on E-Commerce and Web-Technology, 2000, UK.

[8] Qiming. 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.

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

[10] Common Business Library, http://www.oasis-open.org/cover/cbl.html.

[11] xCBL 2.0, http://www.commerceone.com/xml/cbl/.

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

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

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

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

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

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

[18] Groove.net, www.groove.net/peer.gtml.

[19] 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.

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

[21] M. Koetsier, P. Grefen, J. Vonk, "Contracts for Cross-Organizational Workflow Management", Proc. EC-Web'2000.

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

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

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

[25] 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.

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

[27] PeerMetrics, www.peermetrics.com.

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

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

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

[31] Soap: Simple Object Access Protocol, http://msdn.Microsoft.com/xml/general/soapspec.asp, 2000

[32] UDDI: A new Proposed Standard Delivers on Promise of Internet for Business of All Sizes, http://msdn.Microsoft.com/presspass/features/2000, 2000

[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.