ISeB (2004) 2:111–126
DOI 10.1007/s10257-003-0029-8

Information Systems and *e-Business* Management

How public conversation management integrated with local business process management

Qiming Chen, Meichun Hsu, Vinkesh Mehta

Commerce One Inc. 4440 Rosewood Drive, Pleasanton, CA 94588, USA
(e-mail: {qiming.chen;meichun.hsu;mehta.vinkesh}@commerceone.com)

Abstract. In order for enterprises to collaborate at the business-process level, they must deal with two kinds of processes: the public conversation processes specifying inter-enterprise document flows, and the private business processes specifying local workflows of document manipulation and other related tasks. The provisioning, interaction and integration of conversation management and business process management, have become the common interest of the e-business industry. In this paper we discuss the relationship and interaction between conversation management and business process management; point out the difference between public conversation processes (e.g. BPSS processes) and peer-conversation processes (e.g. BPEL4WS processes). We then illustrate our collaborative process management system that has functionally separated conversation manager and business process manager. The conversation manager is based on the ebXML BPSS standard; it is used for validating document exchange at run-time and for activating corresponding process tasks. We have also proposed the conversation model driven asynchronous task activation mechanism for interaction between a conversation process and the coupled business process dynamically. With this mechanism, generic APIs between the conversation manager and the business process manager can be easily defined and used by multiple plugged-in conversation managers.

1 Introduction

1.1 Conversation as abstract interface of business collaboration

In order for enterprises to collaborate at the business-process level, they must allow the business processes run on their local sites to interact [5, 6, 9]. In order to provide abstract interfaces of business interaction distinguished from the concrete services, *Inter-enterprise Conversation Process* (ICP) specification standards such as ebXML BPSS (Business Process Specification Schema) [11, 14], and conversation enabled business process specification standards such as BPEL4WS (Business Process

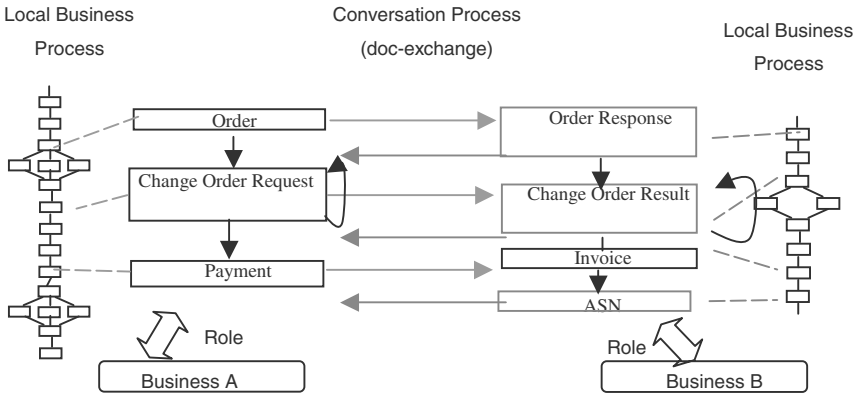

Fig. 1. Choreographed document exchange as conversation process

Execution Language for Web Services) [2], WSCI (Web Service Choreography Interface) [21], WSCL (Web Service Conversation Language) [19] have been proposed. The conversation processes specified in BPSS serve as the common templates for all the participating parties. The conversation processes specified in WSCI or WSCL describe one-party's way of business interactions which is expected to be followed by the peer parties while doing business with it.

As shown in Fig. 1, an ICP specifies the choreography of document exchange as the abstract interface, leaving the processing and provisioning of documents to local business processes or services. As the abstract interface, one ICP can be supported by a variety of business processes and services with different implementations.

In general, a conversation process and a business process have different underlying models. To clarify, we distinguish the basic operations of conversation processes and business processes by calling them *conversation activities* and *tasks* respectively. A conversation activity has two operations for delivering requesting and responding documents, which may map to one or more local tasks for consuming and producing the documents. A conversation process model such as BPSS [11], is different from the traditional workflow models such as WfMC's reference model [17]. The choreography of conversation activities and the flow of task execution are also semantically different.

Thus, in an inter-enterprise collaboration, each participant needs to deal with two kinds of processes: the *conversation process* specifying public conversation-activities, or "document-flow", and the *local business process* specifying private tasks that fulfill the conversation activities. The choreography of document exchange defined in the conversation process actually indicates the expected behavior of the coupled business process in processing and producing the corresponding documents; however, since the conversation process and the coupled business process have different underlying models and run at different paces, synchronizing their execution is difficult.

1.2 Requirements and hard problems

Several approaches have been investigated for managing business conversation. For example, a single round-trip conversation may be made through WSDL invocation but multiple such "points of conversation" would fail to correlate their semantics at process-level; using workflow messages to carry documents as payloads can support document exchange between the same kind of workflow engines but can hardly support ICP standards. In developing our approach, we start with studying the following basic requirements for supporting inter-enterprise collaboration.

- A Collaborative Process Manager (CPM) must support *choreographed conversation* at process-level rather than "*points of conversation*" at action level; support both conversation activities and local tasks; support *well-established conversation process standards* (e.g. BPSS); and furthermore, provides the extensibility for conforming to *multiple* conversation process standards.
- Architecturally, the conversation manager must be light-weight with no function overlap with the coupled process manager. Their functions must be clearly distinguished.
- Interaction between conversation process and local business process at run-time is necessary because even if the task flow of a business process is consistent with the order of document exchange specified in the coupled conversation process, it is difficult to synchronize the execution of business process with conversation process, especially when the business process involves other private applications and runs at a different pace from the conversation process.

To meet the above requirements, it is necessary to provide a standard based conversation manager separately from a workflow engine. To allow a conversation manager and a workflow engine to inter-operate, it is necessary to deal with two fundamental problems:

- the model difference between a conversation process and a business process, and
- the run-time interaction between them.

Semantically, a *conversation activity* specifies the document-flow between roles, and a *process task* specifies the invocation of an action by a single role. The conversation processes and the business processes under different models have different structures as well. At run-time, a workflow engine *actively* schedules tasks to run, but a conversation manager passively waits for the document exchange events. All these indicate the inadequacy of using a workflow engine to handle standard-based business conversation even if it is facilitated with document sending and receiving capability. Further, because a conversation process is essentially modeled as a graph with branching, nesting and looping, it may not be formally partial ordered or mapped to a partial order of tasks under a different process model.

Especially, it is difficult to synchronize a conversation process and the corresponding local business process because a local business process may not be carried out at the same pace as the conversation process, and potentially involves additional private tasks to those related to the

conversation. For instance, while sending a document may be pre-scheduled in the same way as other tasks, receiving a document may not.

In developing a Collaborative Process Manager (CPM) for supporting inter-enterprise process collaboration, we focused on these key issues. Our experience reveals that when these issues are addressed properly, then existing process management platforms can be easily extended to support inter-enterprise collaboration.

1.3 The solution

We have developed a CPM based on the CommerceOne business process management platform to support inter-enterprise business collaboration. Our solutions are characterized by the following.

- Providing the *BPSS-based conversation manager* for *run-time support* of inter-enterprise collaboration. We recognize that business processes specified in BPEL4WS and WSCI are not pure conversation processes but having public interface and local context mixed; and map them to BPSS specification to underlie run-time conversation support.
- Functionally separating *conversation management*(*CM*), *process management* (*PM*) and *action management* (*AM*).
- Having conversation manager and process manager interact through *conversation model driven asynchronous task activation*.

With the proposed *conversation model driven asynchronous task activation* approach, the document exchange information verified by the conversation manager is used to instantiate the conditions for activating local process tasks for document manipulation. This mechanism provides a way for separate CM and PM systems to interact, making their interface fairly dynamic.

The proposed approach allows us to make full use of existing workflow system components, to support both public conversation processes and local business processes, and to interface conversation management and process management dynamically. This architecture also offers the potential of supporting multiple ICP standards by plugging in multiple conversation managers built on the corresponding ICP models.

Section 2 compares public and peer conversation process specifications, and describes how to derive public conversation process out of local business process. Section 3 illustrates the architecture of our collaborative process management system, and discusses the dynamic interaction of public conversation management and local business process management. Section 4 discusses transaction semantics. Finally, some remarks and the comparison with related work are presented in Section 5.

2 Design-time support: extract public conversation process from local business process

To explain the reason for us to deal with the *public conversation processes* specified in BPSS, we first discuss the difference and relationship of a BPSS process and a "*peer conversation process*" specified in BPEL4WS or WSCI.

BPSS is a standard XML language for specifying inter-enterprise, choreographed conversations. In BPSS, a conversation process is called a

collaboration. A *binary collaboration* has two *authorized-roles* and a *multi-party collaboration* has more than two *partner-roles*. The *business partners* participating in a collaboration process play these roles; they interact through a set of choreographed *conversation activities* (called business activities in BPSS). A conversation activity may represent a *business transaction* consisting of one or two predefined *business document* flows between the participating roles. Iteratively a conversation activity may also represent a nested binary collaboration. While a BPSS process specifies choreographed document exchanges, it leaves document provisioning and processing, as well as the decisions on document exchanges, to the local business process at every participant's side.

From this point of view, the BPSS model is a "pure" conversation process model; it deals with messaging only but without incorporating local context of the participants; it provides the abstract interfaces of business interaction, regardless of the concrete implementation.

2.1 Public and peer view to conversation

Unlike BPSS, that specifies conversation processes from public view, some other process specification languages such as BPEL4WS and WSCI, offer the peer-view of conversation. This can be further explained below.

- A *public conversation process* provides a public specification of inter-enterprise collaboration as the legal sequence of document flow, regardless of any local business logic for the documents to be chosen, sent or received. For example, a buyer may either confirm or cancel a purchase order based on some local decision, but the seller is only interested in whether the document received from the buyer is for confirmation or for cancellation, regardless of the cause of buyer's choice.
- A *peer conversation process* describes peer-side interfaces of conversations, incorporating with *local conditions, rules and policies*. While such local context may provide a base for conversation control, e.g. either confirming or canceling a purchase order, it is insignificant to, and therefore should not be exposed to the partner. The typical standard languages for specifying local conversation control are BPEL4WS and WSCI, and the typical use of these languages is to specify process-oriented *service composition* that incorporates conversation activities.

In summary, although BPSS and BPEL4WS or WSCI have the similar expressive power for specifying the choreography of document exchanges, their focus is different in the following aspects.

- The focus of BPSS is on *abstract conversation specification*, leaving the document manipulation to other local business processes; while the focus of BPEL4WS is on *executable process specification* mixed with conversational and non-conversational activites, for that local context has to be taken into account.
- The focus of BPSS is on providing commonly agreed conversation-driven *collaboration protocol*; while the focus of BPEL4WS or WSCI is on specifying one-side conversation-driven *service composition*.

With the focus on specifying collaboration, BPSS is designed in a totally symmetric manner, such that the business service interface of each

participant can be configured from the same collaboration definition that is commonly agreed on. Business collaboration is peer-to-peer and cannot be described by taking the point of view of one-side; it must be specified without any particular party's point of view as a pure message flow. The interface is secondary, almost insignificant, since it can be derived from the message flow.

It is worth noting that the global models used in WSFL [21] (replaced by BPEL4WS) and WSCI do not guarantee that the interfaces specified at the individual collaboration partners' sides are inter-operatable. A global model is merely a one-to-one mapping between port types from WSDL point of view, at the action-level rather than the process-level. Thus compared with WSCI, WSFL, etc, BPSS provides independent collaboration definitions rather than two or more interfaces with a behavior glued together with a global model.

Another difference between BPSS and BPEL4WS or WSCI is the interaction style. WSCI by its name is an interface language for describing conversation flows. Interface based specification is suitable for providing server-centered view but not peer-to-peer view. Business collaboration is peer-to-peer where interfaces disappear behind collaboration definitions.

To sum-up, public and peer conversation processes take different views on conversation and have different roles in business interaction. A BPSS specification provides an abstract and publicly observed view of collaboration, free of the context of any participating party. A BPEL4WS or WSCI specification provides a one-side view of conversation, which may be sensitive to local context. Such a one-side view can be registered as an interface for local service composition.

Figure 2 illustrates such an example where the buyer side service controls a conditional branch of activities for confirming or canceling a purchase order, in the local context. The local interface of conversation in WSCI at the buyer's side specifies such a "complex activity" in the following way.

```
<switch>
<case>
  <condition> tns:CancelOrderCondition </condition>
  <action name = "CancelOrder"
      role = "tns:buyer"
      operation = "tns: BuyerToSeller/CancelOrder"/>
</case>
<default>
  <action name = "ConfirmOrder"
      role = "tns:buyer"
      operation = "tns: BuyerToSeller/ConfirmOrder"/>
</default>
</switch>
```

where the operation of an action refers to a WSDL operation. For example, the operation "BuyerToSeller/ConfirmOrder" is specified as below.

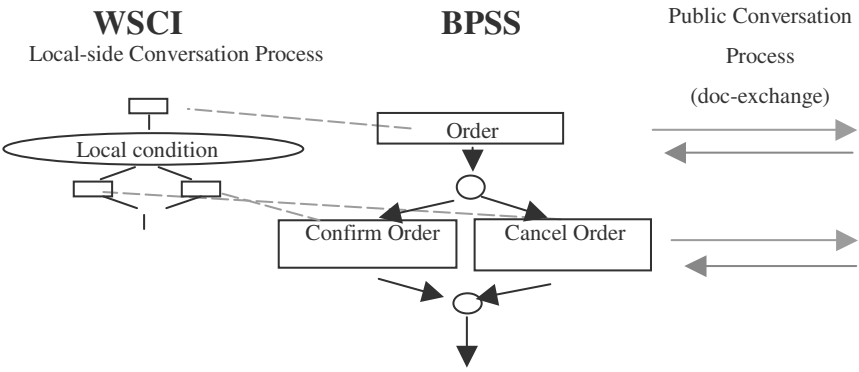

Fig. 2. Manage conversation in local context and in public context

```
<portType name = "BuyerToSeller">
 <operation name = "ConfirmOrder">
  <output message = "tns:ConfirmOrderRequest"/>
  <input message = "tns:Invoice"/>
 </operation>

 <operation name = "CancelOrder">
  <output message = "tns:CancelOrderRequest"/>
 </operation>
</portType>
```

The condition "CancelOrderCondition" is a WSCI property defined in the local context.

However, the local context of one party has no interest to its peer party. As mentioned above, a seller is not interested in why a purchase order is confirmed or cancelled by the buyer, but only interested in which message is actually received.

The public interface of conversation for the above example, expressed in BPSS, looks like the following.

```
< BusinessTransactionActivity
     name = "Confirm"
     businessTransaction = "ConfirmOrder"
     fromAuthorizedRole = "buyer"
     isConcurrent = "false"
     toAuthorizedRole = "seller"/>

<BusinessTransactionActivity
     name = "Cancel"
```

```
        businessTransaction = "CancelOrder"
        fromAuthorizedRole = "buyer"
        isConcurrent = "false"
        toAuthorizedRole = "seller"/>

    <Fork name = "Fork"/>

    <Join name = "Join" waitForAll = "false"/>

    <Transition conditionGuard = "Success"
        fromBusinessState = "Fork"
        toBusinessState = "Confirm"/>

    <Transition conditionGuard = "Success"
        fromBusinessState = "Fork"
        toBusinessState = "Cancel"/>

    <Transition
        fromBusinessState = "Confirm"
        toBusinessState = "Join"/>

    <Transition
        fromBusinessState = "Cancel"
        toBusinessState = "Join"/>
```

where a BusinessTransaction is a WSDL style operation. As an example, with BPSS, the BusinessTransaction "ConfirmOrder" is specified by the following.

```
    <BusinessTransaction
        name = "ConfirmOrder" pattern = "RequestResponse">

      <RequestingBusinessActivity
        name = "ConfirmOrderRequest"
        isAuthorizationRequired = "true"
        isIntelligibleCheckRequired = "false"
        isNonRepudiationRequired = "false">
       <DocumentEnvelope
            businessDocument = "OrderConfirmation"/>

      <RespondingBusinessActivity
```

```
        name = "ConfirmOrderResponse"
        isAuthorizationRequired = "true"
        isIntelligibleCheckRequired = "false"
        isNonRepudiationReceiptRequired = "false">
      <DocumentEnvelope businessDocument = "Invoice"/>
    </RespondingBusinessActivity>
  </BusinessTransaction>
```

This kind of public definition of conversation provides a common view to business interaction; it is free of any local context of any particular participant; it shows the commonly observable behavior of collaboration.

2.2 Mapping BPEL4WS or WSCI to BPSS

In order to handle pure conversation management separately from local business process management, we can

- map a BPEL4WS or WSCI specification to the counterpart BPSS specification, then
- use the derived BPSS specification to underlie conversation management, and
- integrate the conversation management with the business process management that directly or indirectly supports the implementation of the BPEL4WS or WSCI process.

When deriving a BPSS specification out of a BPEL4WS or WSCI specification we keep in mind that the former is used to specify the public observable behavior of conversation, and the latter is used to specify the local control of conversation,. Therefore the major effect of derivation is to drop local context as illustrated in the following examples.

- A conditional branch of activities specified in WSCI, show up in the resulting BPSS specification as a disjunction of activities, regardless of any local business logic behind the selection.
- In BPEL4WS and WSCI, the notions called property and context are introduced to refer to local variables that correlate to message parts, information extracted from messages using "selector", or information about local services. These notions are somewhat related to the local context and have no counterpart in BPSS.

We used XSLT run on Xalan to specify the mapping from a peer conversation process specification in BPEL4WS or WSCI to a public conversation process specification in BPSS. The detail patterns of the mapping will be reported separately.

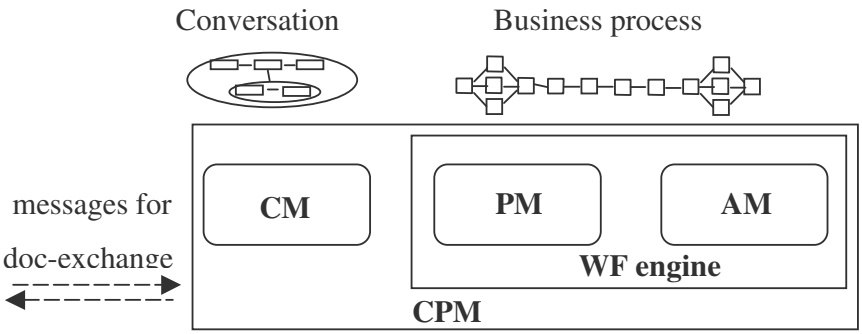

Fig. 3. Conceptual components of a CPM – conversation manager, process manager and action manager

3 Run-time support: dynamic interaction between conversation management and process management

We have developed a CM system under the ebXML's BPSS standard for handling peer-to-peer collaboration between two or more parties, and embedded it in our Collaborative Process Management (CPM) platform [13].

As shown in Fig. 3, the CPM for supporting inter-enterprise collaboration is potentially made of three communicating components for handling conversations, local business processes and actions respectively.

- The Conversation Manager (CM) handles inter-enterprise business interaction based on an *ICP model* (*conversation model*). A core function of CM is to enforce the choreography of document exchange.
- The Process Manager (PM) runs local processes based on a *business process model*. A core function of PM is to enforce the rules for triggering tasks. These tasks contribute to the accomplishment of the local process, including document manipulation as required by the conversation activities.
- The Action Manager (AM) dispatches and invokes local applications, services or processes to perform process tasks. An action provides an actual implementation of document processing, provisioning or other applications. Actions may be called through local or remote invocation, based on CORBA, WSDL, etc. The invocation may be made synchronous or asynchronous.

This architecture provides a clear separation of CM, PM and AM functionally, and allows the maximal usability of existing workflow system components. Run-time conversation support includes the following two major functions:

- Validating document exchange choreography; and
- Enabling CM and PM to inter-operate based on conversation model driven asynchronous task activation.

The BPSS-based conversation manager handles conversation activities similar to the way a workflow system handles process tasks. However, enforcing the choreography of conversation activities must take into account

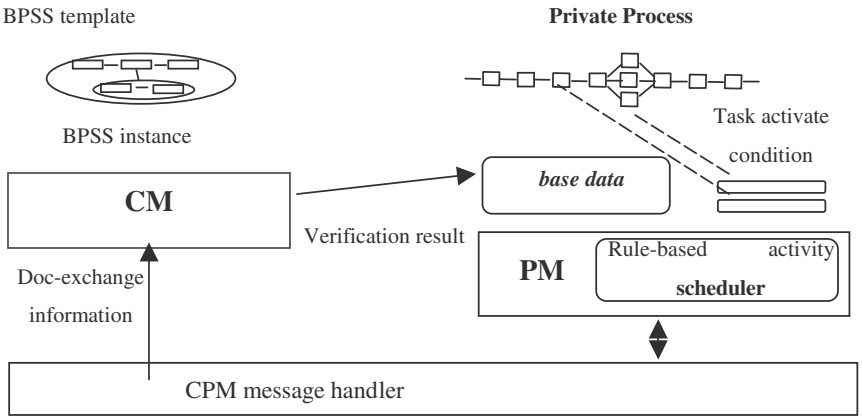

Fig. 4. CM as conversation model driven asynchronous task activator

not only the existence of inbound and outbound documents, but also the order and time for those documents to send or to be received in the interaction with peers. In inter-enterprise collaboration, each enterprise is supposed to provide its own conversation management based on the commonly agreed BPSS specifications.

Given a conversation process, C, and the coupled local business process, P, even if the task flow of P is consistent with the order of document exchange specified in C, it is difficult to synchronize the *execution paces* of P and C in terms of synchronous task activation, especially when P involves other private applications and runs at a different pace from C. In order to resolve this problem, we have developed the *asynchronous task activation* mechanism for managing interaction between a conversation process instance and the corresponding local business process instance.

In a workflow system, activating a task may mean executing it right away or schedule it to run. Conceptually a task is activated by the satisfaction of the "*task activation conditions*". When a process is specified with inter-linked tasks, a link from task T_1 to task T_2 actually represents an activation condition of T_2 relating to the execution status of T_1. From this point, a business process may also be viewed as a set of rules for task activation, process termination, etc. The CPM system developed at CommerceOne Inc. [13], like many others, supports this model.

Further, we distinguish *synchronous* and *asynchronous* task activation. In general, *synchronous activation* means an *event*, such as notifying the status of a precedent task, directly activates the task; *asynchronous activation* means an event causes the update of the base data underlying the task activation condition, which may potentially make the task ready to run, but checking conditions and activating the task is handled by a separate thread of control.

Refer to Fig. 4, the proposed *asynchronous task activation* mechanism can be explained as below.

– *In the execution of a business process*, a task can be scheduled to run based on certain task-activation conditions, and these conditions are checked against certain underlying data.

– *In evolving a conversation process instance* upon sending or receiving a document, valid document exchange information can be generated for updating the "*base-data*" underlying task activation-conditions.

Asynchronously, the task scheduler of a PM will check those conditions to schedule tasks, as a separate thread of control.

At a business site, BPSS based conversation validation is made with respect to each document sent or received. The following *generic document-exchange information* is used as the input parameters of the validation: collaboration ID; conversation-activity name that represents a service; sender; receiver; and document name.

The above document exchange information is validated based on the template and execution instance of the BPSS conversation process. When it is validated, at the minimum the following information will be derived for updating the "*base-data*" underlying task activation-conditions; otherwise appropriate error messages will be returned. The base-data updates can be made by the CM using PM's API, or by the PM using the information returned from CM: collaboration ID; conversation activity; interaction-time; requesting player and its role; responding player and its role; document name; action-type ("response" or "request"); validation status. This resulting data may be selected to underlie task-activation conditions.

It is worth noting that the mapping between the information generated by CM and the base data underlying the task activation conditions in PM must be provided. With such mapping, conversation processes and local business processes can be defined independently.

Interoperating a CM and a PM under the *asynchronous task activation mechanism* actually represents the simplest approach to bridging a conversation model and a business process model. We see that when the BPSS-based conversation manager verifies document exchanges, and has the verification results used to set up the task activation conditions for the local processes,

– the BPSS-based conversation manager may be viewed as the extension of the PM's rules engine for task scheduling;
– the conversation process instance may be viewed as the extension of the "fact base" searched by the rules engine through APIs.

From this point of view, the BPSS-based conversation manager can be considered as a *conversation model driven asynchronous task activator* that the rule based task scheduler must go through. More significantly, by providing a set of generic APIs between CM and PM, multiple CMs can be utilized for supporting different inter-enterprise interaction standards, and interact with PM in a uniform way.

The above Architecture provides a clean separation of Conversation Process Management and Local Process Management. Alternatively, we can also treat Conversation Manager as a special Action Manager. Let's call this special Action Manager for Managing Conversation the *Conversation Action Manager(CAM)*.

Action Managers deal with executing *tasks*. The *Conversation Activities* of a Conversation Process can be modeled as a set of special *task*. There are primarily four kinds of *Conversation Activities* or *Conversation Tasks*. Send document, Receive Document, Send and Receive Document, and Send Response to a Received Document. These map to the WSDL Notify, One

Way, Request Response and Solicit Response (The response portion). The CAM executes these tasks.

Normally an Action Manager just manages execution of *tasks,* but CAM is a special Action Manager it goes beyond execution of individual tasks, it also manages the overall Conversation Process associated with these *tasks*. The CAM evolves and validates Conversation Processes as individual Conversation Tasks get executed.

The CAM supports the following features:
– Provide interface for PM to trigger *Conversation tasks*.
– Interface with Messaging layer for performing document exchange
– Evolve and Validate Conversation Process

By combining the features of Action Manager and Conversation Manager into one, the cost of handling each message is reduced. Each message is opened and parsed just once. Also there are no new interfaces that need to be exposed by PM. The CAM uses the standard interface exposed by PM for AM interaction. This Architecture provides a better overall performance and cost of maintenance.

4 Supporting transaction semantics and preventing deadlocks

4.1 Supporting transaction semantics

As business collaboration involves multiple participants without centralized control, in order to support transaction semantics, appropriate collaboration protocols must be introduced and followed by each participant on a trustable basis. Further, messages exchanged between participants must carry the information, indicating which collaboration protocols are being adapted.

From transaction point of view, there exist two types of conversation oriented inter-business collaborations.

– Transactional collaboration where the conversational activities have a short duration and executed within limited trust domains. Their collaboration has the ''all or nothing'' property; but which is enforced at each participant's site individually according to the protocols they supposed to follow.
– Non-transactional collaboration where the conversational activities are long in duration and desire to apply business logic to handle business exceptions. The long duration prohibits locking data resources to make actions tentative and hidden from other applications. Instead, actions are applied immediately and are permanent. In case of failure, each participant can make compensation at its own site at its own choice.

The protocols of transactional collaboration govern the atomicity of multiple conversation activities and the associated process activities. Existing transaction processing systems can be used at each site. The transaction related Information exchanged between collaboration participants include the status with respect to two-phase-commit, commit/about decision, etc. We plan to adapt the notification mechanism described in WS-Coordination

[22] and WS-Transaction [23] proposals to synchronize peer-wise transactionmanagement.

4.2 Preventing deadlock caused by message delivery mis-order

A key design issue is to maintain the right order of message processing. For various reasons the messages may not be delivered in the expected order, for example, consider a process collaboration involving three peers 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.

- When task T_1 run at A completed, A notified B and C the completion of T_1, in a message, msg_1;
- Upon receipt of msg_1, B started executing task T_2;
- When T_2 completed, B notified A and C the completion of task T_2, in a message, msg_2.

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 C can lead to an inconsistent result.

Queuing technique and the knowledge drawn from conversation process definitions are used to resolve the out-of-order message delivery problem. Each site is facilitated with a conversation process specific queue interfaced to the process definition handler and the process instance log handler, using process definitions and execution histories to make operational decisions.

The conversation manager is provided with a queue manager with the following functionalities.

- When a message is received, check if it is ready to be processed based on the conversation 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 C since C hasn't received the task completion 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 C queued msg_2 for task T_2 since it did not receive the task completion message, msg_1, for T_1. Later, when msg_1 was eventually received, C would process msg_1 for T_1 first, followed by processing msg_2 for task T_2.
- When an 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.

5 Conclusion and comparison

In order for enterprises to collaborate at the business-process level, they must support two kinds of processes: the public conversation process specifying the ''conversation-flow'', and the local business process specifying the ''workflow'' that fulfills the conversation activities.

In this paper we discussed the relationship between conversation management and business process management, and between public conversation processes and peer conversation processes. We then illustrated the system we built for inter-enterprise collaboration that is characterized by separating conversation management, business process management and action management, and by interacting a conversation process instance with a business process instance dynamically in terms of the *conversation model driven asynchronous task activation* mechanism. The conversation manager can be used to validate document exchange at run-time, to activate process tasks for manipulating documents, and to test the behavior of local processes in supporting conversations.

Our approach is based on peer-to-peer process interaction model, it clearly differs from the centralized workflow management [21], from the conventional process inter-operation for enforcing ad-hoc task dependencies and data exchanges in a single enterprise, and from the invocation based process decentralization seen in [12], etc. This work has also elevated the agent based peer-to-peer interaction [6–7] to the process level.

Different from WSDL [20], Rosetta-net [15], BPML [3], that support point of conversation at action-level but not directly correlated at the process-level, this work focuses on choreographed conversation. Indeed, dealing with point of conversation can provide certain flexibility, but can hardly follow a commonly agreed conversation model standard such as ebXML BPSS [11]. Further, BPEL4WS [2] and WSCL [19] are used to offer a *single party view* rather than the *public view*, to the collaboration. As a result, an implementation does not present a general model of peer-to-peer synchronized execution among multiple conversation activities; for instance, it does not intend to address how the partner process instances are synchronized, or made aware of the progress of the peer processes.

Interoperating inter-enterprise collaborations and intra-enterprise business processes is a very practical and challenging issue faced by many organizations. We see the difference between conversation models that underlie the ICP standards and the conventional business process models that the existing workflow engines support. Most of the current efforts are characterized by adopting one kind of models, either to ''simulate'' conversation activities by the business process tasks, or take local processes as ''point of services'' to ''fulfill' conversation activities [1, 3, 20]. There lacks a formal execution mechanism for interacting the public conversation process execution and local business process execution at run-time. In fact, to our knowledge, there is no real implementation reported by far on separating run-time conversation management and business process management, while interacting them dynamically.

Compared with the related work, our approach allows us to provide a clear separation of inter-enterprise conversation management and local business process management, to make full use of existing workflow system components, to support both public conversation processes and local business processes, and to interact CM and PM dynamically.

We are currently studying the mechanism for modeling both interface-based service composition and inter-enterprise collaboration, towards the development of a unified specification language.

References

1. BEA System(2002) Intalio, SAP, Sun Microsystems, "Web Service Choreography Interface"
2. BPEL4WS, "Business Process Execution Language for Web Service", www-3.ibm.com/software/solutions/webservices/
3. BPML (2002) "Business Process Markup Language", www.BPMI.org
4. Chen Q, Hsu M (2002) *CPM Revisited – An Architecture Comparison*, 2002 IFCIS Conference on Cooperative Information Systems (CoopIS'2002), USA
5. Chen Q, Hsu M (2001) *Inter-Enterprise Collaborative Business Process Management*, Proceedings of 17th International Conference on Data Engineering (ICDE-2001), Germany
6. Chen Q, Dayal U, Hsu M (2001) *Conceptual Modeling for Collaborative E-business Processes*, ER-2001
7. Chweh CR (2001) *Peer-to-peer computing transforms file-sharing and large-scale distributed computing*, IEEE Software, Vol 18, No 1
8. Clark D (2001) *Face-to-face with peer-to-peer networking*, Computer, Vol 34, No 1
9. Dayal U, Hsu M, Ladin R (2001) *Business Process Coordination: State of the Art, Trends, and Open Issues*, Presentation on VLDB 10 years best paper award, Proceedings of VLDB 2001, Italy
10. Document Object Model, http://www.w3.org/DOM/
11. EbXML.org, Business Process Specification Schema", V1.01, 2001
12. Koetsier M, Grefen P, Vonk J (2000) "Contracts for Cross-Organizational Workflow Management", Proc. EC-Web'2000
13. Mehta VO, Shrotriya S (2001) "Collaborative Business Process Management System Update for Marketsite Version 4.6", Internal Document, Commerce One Inc.
14. OAG, "OAGIS Implementation using CPA, CPP and BPSS Specification 1.0", http://xml.coverpages.org/OAGI-ebXML-WhitePaper-103.pdf
15. Rosetta-net, www.rosettaNet.org
16. SOAP, "Simple Object Access Protocol", http://msdn.Microsoft.com/xml/general/soap-spec.asp, www.w3c.org
17. Workflow Management Coalition, www.aiim.org/wfmc/mainframe.htm
18. WSCI, "Web Service Choreography Interface", Tech Report by Italio, SAP, BEA, Sun Microsystems. 2002
19. WSCL, "Web Service Conversation Language", HP Submission to W3C, www.w3c.org
20. WSDL, "Web Service Description Language", www.w3c.org
21. WSFL, "Web Service Flow Language", www-3.ibm.com/software/solutions/webservices/
22. WS-Coordination, http://www-106.ibm.com/developerworks/library/ws-coor/
23. WS-Transaction, http://www-106.ibm.com/developerworks/webservices/library/ws-tran-spec/