Conversation Support for Business Process Integration

James E. Hanson, Prabir Nandi, Santhosh Kumaran
IBM T.J. Watson Research Center
Yorktown Heights NY 10598
{jehanson | prabir | sbk}@us.ibm.com

Abstract

Business Process Integration and Automation (BPIA) has emerged as an important aspect of the enterprise computing landscape. Intra-enterprise application integration (EAI) as well as the inter enterprise integration (B2B) are increasingly being performed in the context of business processes. The integration scenarios typically involve distributed systems that are autonomous to some degree. From the BPIA perspective, the autonomy refers to the fact that the systems being integrated have their own process choreography engines and execute internal business processes that are private to them. In the case of B2B integration, the systems being integrated are fully autonomous, while various levels of autonomy exist in systems partaking in EAI.

In this paper we present a new paradigm for business process integration. Our approach is based on a conversation model that enables autonomous, distributed BPM (Business Process Management) modules to integrate and collaborate. Our conversation model supports the exchange of multiple correlated messages with arbitrary sequencing constraints and covers the formatting of messages that are to be sent as well as the parsing of the messages that have been received. The crux of our conversation model is the notion of a conversation policy, which is a machine-readable specification of a pattern of message exchange in a conversation. Our model supports nesting and composition of conversation policies to provide a dynamic, adaptable, incremental, open-ended, and extensible mechanism for business process integration.

We discuss the current implementation of this conversation model and early experience in applying the model to solve customer problems. The implementation utilizes distributed object technology.

1 Introduction

As business applications become more complex, and application-to-application integration takes on greater importance, we are seeing the emergence of business process integration (BPI) as a key requirement in enterprise computing systems. Tightly coupled solutions (e.g., CORBA) have difficulty in a heterogeneous, dynamic environment. Stateless interactions (e.g., XML over HTTP) fail to support the essentially multi-step character of typical business interactions. Protocols tend to be either ad hoc and privately agreed-upon or industry-wide consortium-driven (e.g., RosettaNet). The ad hoc protocols do not scale and need to be reformulated for every trading partner, while the consortium-driven protocols are slow to change and impossible to innovate on.

The aim of this paper is to propose general-purpose conversation support as a solution to the needs of business process integration. The next section is devoted to a general discussion and validation of conversation support, with special attention to the most challenging case, that of B2B integration. Section 3 goes into the details of its core element, conversation policies. Then, in section 4, we turn to the current implementation of a general-purpose conversation support system in a business integration platform called Ninja, with emphasis on its conversation support aspects. Section 5 briefly reviews some related work, and in section 6 we close with a preview of our plans for further development.

2 The conversational model of component interaction

Conversation support for business process integration starts with the adoption of the conversational model of component interactions, in which components (applications, e-businesses, agents) are treated as autonomous, loosely coupled entities which interact by exchanging messages in a conversational context.

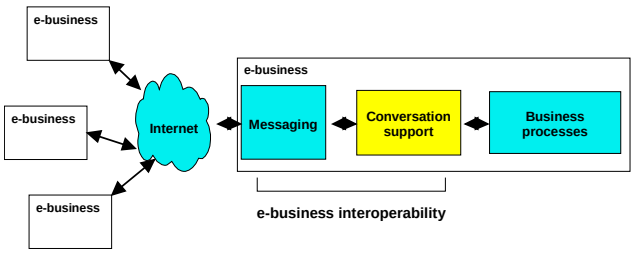

Figure 1: Conversational model for B2B

Figure 1 shows the high-level architectural elements of the conversational model, as applied to business-to-business interactions. In the conversational model the actual process logic is fronted by interoperability technology specifically devoted to managing interactions. The interoperability technology consists of two distinct parts: messaging and conversation support. Messaging is the bare “plumbing” needed to send and receive electronic communications with others. Conversation support governs the formatting of messages that are to be sent, the parsing of messages that have been received, and the sequencing and timing constraints on exchanges of multiple, correlated messages. Conversation support is a separate subsystem that mediates between the messaging system and the business processes.

At this level of detain, the conversational model has much in common with the approach taken by tpaML[2] and ebXML[3]. As we will see, however, it supports a much greater degree of flexibility and expressiveness. This is discussed further in Section 5.2.

The architecture provides a number of features desirable for business process integration. These features are discussed below in detail.

2.1 Interaction via asynchronous message exchange

This means that, instead of exposing a means by which other components can invoke its functionality, a component exposes a means by which others can send it messages. Messages are in effect requests, to be processed as the recipient sees fit. Replies, if any, are made asynchronously by sending another message back.

Message exchange has an important advantage over interaction via direct invocation of functionality: it correctly describes the component’s true control boundaries. This is especially clear in B2B scenarios. For example, if an e-business directly exposes its RFQ processing functionality as a service to be invoked by its customers, that implies that it’s the customer who causes the RFQ to be processed. Really, of course, the firm inserts some control logic into the code that gets invoked, whereby the firm makes the decision of whether to really process the RFQ by calculating a quote and sending it back, or whether to refuse the customer’s request. This control point changes the entire meaning of the interaction. It means that what the customer actually does is submit an RFQ with an implicit request that it be processed--i.e., the customer sends a message. The existence of the control point converts the “service invocation” into a “message delivery”.

Adopting a message-exchange model from the outset makes the real nature of these interactions explicit.

2.2 Generic messaging

Generic messaging means that the message-delivery middleware does not filter the message content, nor constrain the set of messages that may be delivered. It means that arbitrary message content may be exchanged by two parties in a conversation, even in cases where the recipient of a message is unable to recognize its meaning, make decisions about it, or even, perhaps, parse it. In effect, all messages in a conversation are delivered unopened to the same “inbox”, regardless of content.

Generic messaging is an essential element in achieving loose coupling between components, and thereby avoiding the serious integration problems of strongly typed schemes such as CORBA.

Generic messaging is also a proper assignment of function. Placing prior constraints on the set of messages that may be sent or received is like programming your telephone to send or receive only words spoken in English (if such a thing were practical). It is a basic misplacement of function. The proper "job" of the messaging infrastructure is to deliver messages--not to act as a supervisor defining what may and may not be said in a message. As we will see, that job is properly assigned to the conversation support.

In fact, "unexpected" messages may turn out to be valuable, both as feedback about the way the sender wants to interact—e.g., its preferred protocols--and because unfamiliarly-structured messages may well contain clues as to how they should be handled. For this reason, they should be available for inspection and processing by the business logic, not filtered out by a restrictive messaging system. By analogy, if you answer the phone and hear a voice that sounds like it might be speaking in French, you might try to find someone nearby who could serve as interpreter; or, failing that, you could reply in English and hope the other party will start speaking your language. But either case would be preferable to having a phone that refused to receive non-English messages.

2.3 Conversation-centric interactions

At least as important as the adoption of generic asynchronous message exchange is the adoption of "conversation-centric" interactions. This means that messages are sent within an explicit conversational context that is set up on first contact, maintained for the duration of the conversation, and torn down at the end. Each new message in a conversation is interpreted in

relation to the messages previously exchanged in that conversation.

Adopting conversation-centric interaction from the outset amounts to recognizing that in the real world of business process integration, interactions are typically, and most naturally, represented as multi-step exchanges of correlated messages.

2.4 Conversation management independent of message delivery

As we said above, the messaging subsystem encapsulates the sending and receiving of messages, making it possible to support multiple transport mechanisms (e.g., XML over HTTP, SOAP, JMS, etc.) by simply plugging them in.

Furthermore, by separating conversation management from message delivery, it becomes easy to switch delivery mechanisms in mid-conversation—e.g., increase or decrease the encryption level, change to a channel with higher or lower bandwidth, etc. Such a change would itself be negotiated by means of a short sub-conversation embedded in the larger conversational interaction, but would otherwise not impinge upon the interaction.

2.5 Isolation of interoperability from business process

The main reason for making interoperability technology separate from the business process logic is that the interoperability technology shouldn't place constraints on how the core of the business works. The business processes are what the interoperability technology is supposed to support, not prescribe. They are the thing that differentiates one firm from another; the thing that is most crucial to success and survival; and not the kind of thing a firm would like to expose to the world. Interoperability means connecting up the business processes with the economy--not turning the business over to someone else.

Controlling the business processes is the core of what it means to be an independent business engaged in trade. Each party in a trade, by definition, makes decisions unilaterally and executes them under its own control. Even when under contract, a firm's "sovereignty" is not compromised, because its decision to obey the contract is unilateral (as, of course, was its decision to sign the contract in the first place). To the extent that "interoperability" comes to encompass a firm's decision-making and/or execution processes, that firm is not engaging in trade--it is obeying directives.

Furthermore, it is futile to try. From outside, there is no way to tell for sure whether a firm is "unable" to execute a purchase order (for example), or "unwilling" to do so.

Other considerations:

- Business processes change on different timescales from interoperability technology. Changing a business process needs to be done at a firm’s instigation, on the firm's own timescale. It should not be dependent on its customers, suppliers, and trading partners. Changes in interoperability technology are, by definition, on a "shared" timescale.
- Ease of modification. As we will see below, changes in interoperability can be accomplished by something as easy as downloading an XML file. Therefore, changes in business processes are neither forced by changes in interoperability technology, nor hindered by it.

Though interoperability technology and business processes are clearly linked, just as clearly they are separate endeavors with separate driving forces, requirements and timetables.

2.6 Dynamic and flexible model for business process integration

The current models for B2B integration are based on process flow graphs [5],[6],[7],[8],[9]. A company participating in a B2B integration scenario publishes and implements a “public process”. Trading partners of this company communicate with it as stipulated by the public process definition. A process flow graph is used to represent the public process. This approach lacks the flexibility to support dynamic B2B integration. In contrast, our conversational approach presents an incremental, open-ended, dynamic, and personalizable model for B2B integration. We compare and contrast our model with the traditional models in detail in a later section.

3 Conversation policies

In application-to-application conversations, free-form dialogs are not really practical. Therefore, e-business interactions will make frequent use of pre-programmed patterns. Pre-programmed interaction patterns are called conversation policies (CPs). They are central to practical conversation support.

A conversation policy is a machine-readable specification of a pattern of message exchange in a conversation. CPs consist of message schema, sequencing, and timing information. They are conveniently described by a state machine, in which the

sending of a message (in either direction) is a transition from one conversational state to another.

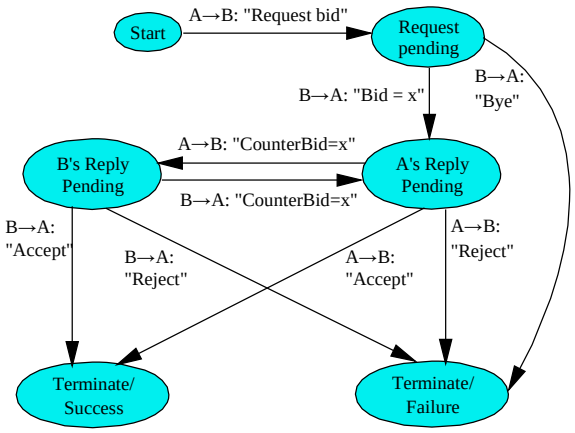

Figure 2: Schematic of a simple CP

Figure 2 shows a schematic of a simple CP, in which two participants, A and B, trade bids and counter-bids until one or the other of them accepts the current bid or gives up. Nodes in the graph correspond to different states of the conversational protocol. In effect, each node represents a summary of what has transpired so far in the conversation. Edges connecting nodes represent transitions from one state to another. Each transition corresponds to a message being sent by one or the other party, and specifies the format or schema of the message as well as which party is the sender. For example, in the starting state (labeled "Start") there is one transition, labeled "A→B: Request Bid", which corresponds to A sending a message to B of the form "Request bid". (Obviously in a real CP, the format of this message would be spelled out.) The CP does not define any other way for the conversation to proceed from its starting state. Similarly, there are two transitions out of the state labeled "Request Pending": one in which party B sends a message to party A of the form "Bid = x" (where x represents some value determined by B), and another in which party B sends "Bye".

In carrying on a conversation, each party separately loads its own copy of the CP, separately maintains its own internal record of the conversation's "current state", and uses the CP to update that state whenever it sends or receives a message. From the point of view of either party, this CP has two types of transitions: transitions to take when a message of a particular format is received, or transitions to take in order to send a message of a particular format. The sender of a message usually (though not always) has to make a decision as to which of the possible alternative messages to send, and often supply data as well--e.g., the value to fill in for bid's amount. Similarly, the recipient usually has to classify the message--identifying which of the possible alternatives was sent--and often parse it to unpack the data supplied by the sender.

As written, the CP is independent of the "point of view" of the company: i.e., which role, A or B, a given company is playing in a given conversation. One can just as easily describe the CP from within one role. For example, if we adopt role A, then transitions labeled "A→B" are interpreted as "send message"; and "B→A" is interpreted as "receive message".

Take this example as an illustration. In a real system, the message schemas would be spelled out in detail, e.g. via reference to an XML Schema document. And, as we will see below, the overall CP could be broken up into "patterns" or "stanzas", each of which was represented by its own CP state machine.

Some other features to note:

- CPs enable extensive reuse of messages. Because a message is interpreted with respect to the conversation's current state, the same message can be safely reused in multiple contexts. For example, the message "OK" can be used in a bid/counterbid CP to signify acceptance of a bid, in an RFQ CP to signify acceptance of a quote, and so forth. In all cases, the contextual information supplied by the CP and the conversation's current state removes any ambiguity.
- CPs provide economy of expression. No need to make messages self-describing "kitchen sinks" containing all possible context you can think of or might ever want to use.
- Because each of the conversing parties maintains its own record of the conversation's state, and uses its own CPs to update that record, the parties need not, in fact, be using exactly the same CP. The minimal requirement is that, in the course of a particular conversation, the sequence of messages they exchange corresponds, on each side, to some path through the particular CP that party is using.

3.1 Nesting and composition of conversation policies

In day-to-day business, a firm's interactions with other firms tend to be made up of common, conventional interaction patterns. That is to say, its conversations tend to have phases or "stanzas" which fall into common patterns, and are reused in different contexts. For example, first there might be discussion of product discovery, then negotiation of the deal, finally settlement. And it is nested: Product discovery, for example, might start with the customer expressing needs,

the seller asking pointed questions about them and then recommending a list of possible matches, followed by the buyer making a selection from the list. Negotiation might start with a discussion of the way to negotiate: haggle over price, or place bids in an auction, or etc., followed by, in both cases, a pattern of message exchange appropriate to that negotiation method. After the products are dealt with, then the parties might turn to a dialog about delivery options (if the goods are physical) and prices. Similar, settlement might start with an enquiry into the methods of payment supported, followed a selection of one of them.

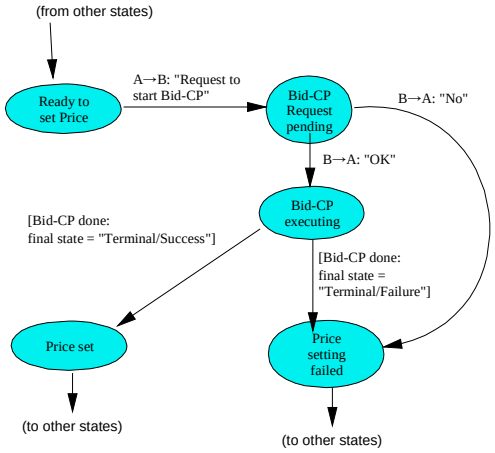

Figure 3: Nested CPs

Conversation policies are inherently nestable. This means that, as part of carrying on a conversation that obeys a given policy, the conversing parties might choose to start a new conversation policy as a "sub-conversation", possibly carry it out to completion, then return to the previous conversation policy. In effect, both parties carry on a narrowly scoped "child" conversation within the enclosing context of the more broadly scoped "parent" conversation.

For example, two parties might be engaged in a simple negotiated bidding procedure, in which they first identify a set of services to be performed, then they engage in an iterated bidding procedure to settle on a price. That iterated bidding procedure may be represented by the CP in Figure 3, in which the specification of the goods outside the scope of the CP--it is part of the context--and the messages only pertain to the bid price.

In this case, the CP governing the "parent" conversation would contain transitions for starting up a sub-conversation following the bid/counterbid CP. This is shown in Figure 3.

4 Conversation support in the Ninja Gateway

Let us now turn to the way in which conversation support is built into a research prototype BPI platform we are developing, called Ninja. Ninja's architecture exemplifies the conversational model, starting with the separation of the interoperability technology from the core business process management technology.

4.1 Ninja Gateway

The Ninja Gateway encapsulates the interoperability technology. It is packaged as a unit capable of operating on its own, or in conjunction with Ninja's Business Process Manager subsystem, the Ninja Process Broker.

The gateway's architecture is indicated in Figure 4. The Connection Manager provides the messaging. The Conversation Adapters provide the conversation support. Each conversation adapter controls one conversation at a time. Additional elements in Figure 4, such as Security and Solution Management, are outside our present scope.

The Connection Manager supports and encapsulates a variety of messaging protocols, such as SOAP, RMI, and plain HTTP. It is designed so that additional protocols may be added as pluggable modules. It supports asymmetric messaging within a conversation--i.e., outbound messages in the conversation sent via one protocol, inbound received via another.

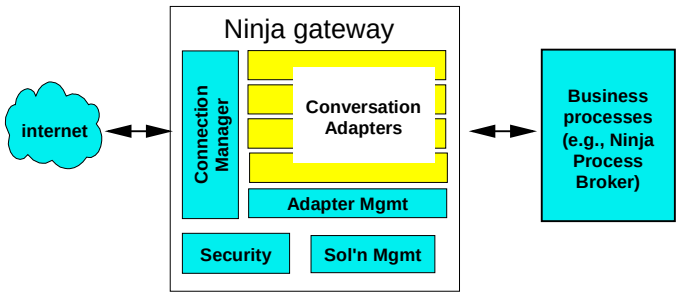

Figure 4: Ninja system overview

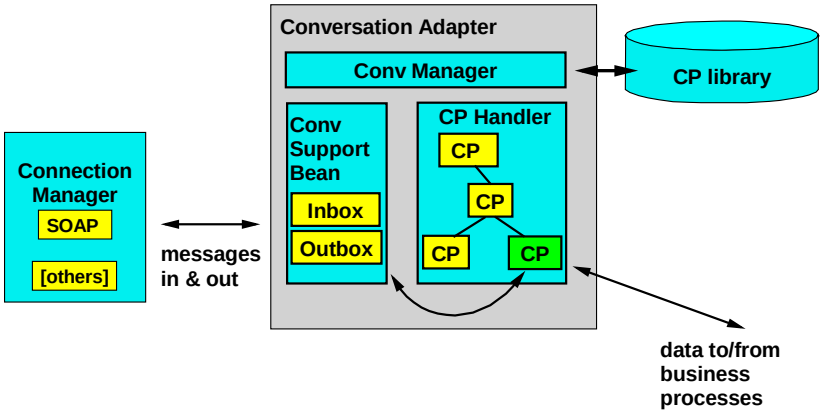

Figure 5: Conversation adapter components

4.2 Conversation Adapter design

A sample Conversation Adapter is shown in Figure 5. It contains a Conversation Support Bean (CSB), a Conversation Policy Handler (CPH), and a Conversation Manager. The CPH holds a tree of CP instances. The Conversation Adapter passes outbound messages to the Connection Manager for delivery, and receives inbound message from it for processing. On the other side, it sends data to, and receives data from, the business processes.

Each Conversation Adapter acts as a dynamic, adaptive channel supporting a single conversation. Multiple simultaneous conversations are handled by separate Conversation Adapter instances. When a conversation is first set up, a new Conversation Manager, CSB and CPH are created, for the purpose of managing that conversation. Typically a CP instance is created as well, and installed as the root of the CPH's tree. Then, as the conversation proceeds, other new CP instances are created and installed in the CPH's tree, as needed. Finally, when the conversation ends, all of these structures are torn down (or pooled for reuse in another conversation).

4.3 Conversation Support Beans

Each Conversation Adapter contains a Conversation Support Bean (CSB) that takes care of maintaining the conversational context. This is straightforward: the CSB consists mainly of an "inbox" into which all incoming messages in the conversation are placed, in order of arrival, and an "outbox" in which outgoing messages are place, for delivery by the connection manager. In a conversation, the outbox of one party is in effect connected to the inbox of the other.

CSBs are created during a conversation setup phase, in which the two parties exchange inbox identifiers. Then, in each subsequent message, the sender uses the recipient's inbox identifier to direct the message to that inbox.

In Ninja, the Connection Manager is the common Internet endpoint for all messages in all conversations that a firm engages in. In order to direct a message to a particular conversation, the sender's connection manager inserts the recipient's conversation-specific inbox identifier into the header of each outgoing message before delivering it. Then, upon delivery, the recipient's connection manager extracts that inbox identifier from the header and uses it to put the message in the inbox of the CSB set up for that particular conversation.

4.4 Conversation Policy Handler

The Conversation Policy Handler (CPH) maintains a set of CP instances in use during the conversation.

CPs are arranged in a tree, which is managed by a CPH. The job of the CPH is to create new nodes in the tree, when an extant node wants to start a "child" conversation; and to delete nodes in the tree once that CP is no longer needed--e.g., when that part of the conversation is over.

One CP in the tree is designated as the Active CP. This is the CP that is currently being used to carry on the conversation. When, during the course of conversing, it comes time to start a sub-conversation, the Conversation Manager creates a new CP instance of the appropriate type, installs it in the CPH as the child of the Active CP, and then makes the newly created CP the new Active CP. When that sub-conversation is over, the Conversation Manager removes it from the tree and makes its parent the Active CP once again.

Starting a sub-conversation is an example of leaving a conversation unfinished (i.e., the parent), carrying on another conversation for a while, and then returning to the unfinished conversation. It is also possible to leave sub-conversations unfinished, return to a higher contextual level (i.e., a higher node in the tree), and start a new CP from that node. This is why the CPH arranges its CP instances in a tree structure, rather than in a stack.

The CP tree provides a certain degree of graceful error handling. Built into the handling of messages is the default behavior that, if a message is received that does not conform to any of the messages allowed by the protocol at that point in the conversation, the message gets passed up to the parent CP, which is then re-activated. If, for example, a CP for processing RFQs receives a message it does not recognize--a query about the shipping information, its default behavior is to pass that message off to its parent. This reflects the fact that the parent, with its broader context, is more likely to recognize the message than the child. If the parent does not recognize it, it passes the message up to its parent, and so forth, all the way up to the root node.

Other ways of handling unexpected messages can be built into individual CPs. For example, a CP may itself have a transition for "none of the above"--i.e., if any message other than the ones expected is received, take that transition. This is appropriate when the sequence of messages needs to be carefully constrained, such as in the middle of a payment, for example.

4.5 CP instances

Each CP consists of a state machine, a set of message formatting or parsing modules, and a set of "command" modules. Formatting modules are used to convert data, such as part numbers, quantities, prices, etc., for sending. Parsing modules do the inverse operation: they unpack a message that has been received. Command modules are calls to the business processes, used when a decision needs to be made and when data must be supplied for formatting an outgoing message.

Processing of an incoming message is as follows:

1. The ConnectionManager places the message in the CSB's inbox, raises a MessageReceived event, and returns a delivery acknowledgement to the sender.
2. The ConversationManager picks up the message and attempts to find a transition to take in the current active CP. It does this by searching for a transition from the CP's current state that corresponds to receiving that particular message. This involves executing a message-parsing module associated with that transition, which compares the format of the message against an expected schema, and, if the format is correct, unpacks the data in the message and places it in a holding area.
3. If such a transition is found, the ConversationManager updates the CP's current state (to the destination state of the transition) and executes any other actions associated with that transition. This will often involve passing the message's data on to the business processes.

Often, as a result an event such as the receipt of a message, the CP moves to a state from which there are transitions for sending messages. This is a decision point in the CP, in the sense that information from the business processes is required in order to select which transition to take, and/or to specify the data to be packed into an outgoing message.

These transitions are taken at the instigation of the business processes. That is, the CP itself does not "call" the business process for a decision, or for data. Rather, the business process raises an event on the CP, which specifies which transition it should take, and supplies the data it should use. In this way, the business processes are always in charge of all outgoing messages.

4.6 Implementation

A detailed description of the implementation is out of the scope of this paper. Here we provide a brief overview of the Ninja gateway implementation.

The Ninja Gateway has been implemented as an application atop a J2EE based platform. We used IBM's Websphere Application Server Advanced Edition as the J2EE platform.

- Conversation Support Beans (CSBs), Conversation Policy instances are implemented as Entity EJBs. The ConversationManager is implemented as a Session EJB.
- Persistence of connection pipes for a conversation instance is done using Entity EJBs while the ConnectionManager itself is implemented as a Session EJB. SOAP and HTTP specific message receivers were deployed as servlets.
- Conversation Policies, defined as a subset of UML state machine, were specified using XML and so were the commands and the command implementations.

5 Related Work

This section discusses the related work in this area.

5.1 Flow Models

Today's BPM integration landscape is dominated by flow models [5],[6],[7],[8],[9],[10]. At the core of this approach is the representation of a set of activities and their dependencies via a directed graph. The nodes in the graph correspond to activities and the arcs represent the

dependencies between these activities. For each inter-enterprise business process, a process flow graph is defined and implemented in the "gateway" of each participating enterprise. These process flow graphs together form the "public process". We list below some of the limitations of this approach:

1. Modeling gap. The flow models only cover a relatively small area of the B2B integration space. In particular, they do not model the life cycle management of these process fragments, process brokering among these fragments, or content aggregation aspects of business process integration problem.
   a. Life cycle management. The creation and termination of each process flow graph is an important aspect of the modeling of a business process integration problem. The application state directly influences the life cycle management.
   b. Process brokering. Any non-trivial business process integration scenario is bound to have multiple, concurrent, and active processes. A business event from a trading partner could influence any of these concurrent processes. Process brokering is concerned with the propagation of business events to the appropriate processes based on the application state[13].
   c. Content aggregation. Typically, the information necessary to make decisions in the context of the execution of a business process is fragmented and distributed. The content aggregation is concerned with dynamically composing the content from multiple enterprise information systems[13].
2. Specification explosion. Any attempt to address the modeling gap problem results in specification explosion as shown in ref. [11]. This implies that the complexity of the flow graph increases exponentially as the flow graph model itself is used to address the problems discussed above.
3. Lack of support for event based programming. Once a flow graph is instantiated, the type of activities in the flow determines the interruptibility of the flow. The flow may be interrupted at an activity waiting for a specific event to arrive. But the model does not support generic event processing. Specifically, the flow models do not support the capability to perform a set of actions based on application state and the event content.
4. Lack of support for dynamic integration. In a dynamic integration scenario, the artifacts that support the integration need to evolve dynamically based on the context. In essence, this amounts to the generation of business process integration "glue" via composition of orthogonal components. Current B2B implementations based on flow models lack this capability.

We list below the distinguishing features of our conversational model and discuss how these features help in addressing the problems mentioned above.

1. Modeling the execution of a conversation policy as a finite state machine.
2. Use of the command design pattern[12] to model the actions a participant needs to perform in response to the receipt of a message in conversation.
3. Modeling of a multi-threaded conversation by dynamic composition of orthogonal conversation policies.
4. Delegation of message parsing and generation to the receivers of the commands and the ability to spawn a child policy based on the parsing of the received message.
5. Ability to manage the life cycle of process flow graphs via commands executed by the finite state machines that model the conversation polices.

The conversation policies can effectively model the life cycles of public processes. They can perform state-dependent brokering of concurrent public processes and dynamically aggregate content for decision support. These are achieved primarily by the use of finite state machines and the command design pattern in modeling the system.

The ability to dynamically compose orthogonal conversation polices eliminates the specification explosion problem.

Our model fully supports event based programming paradigm. A conversational tree supports the processing of a set of events as each node in the tree is modeled as an event-driven state machine. If an event is received that is not in this set, it is propagated to the root of the tree and the root may dynamically spawn a new child policy to process this event.

Our model supports dynamic business process integration by its ability to evolve dynamically based on the process context. For example, in a B2B integration scenario, one could visualize the CP trees at both sides of the conversation growing and shrinking dynamically as the life cycle of a collaborative process unfolds.

5.2 ebXML and tpaML

In many ways, our approach to B2B integration is similar to that taken by tpaML[2] and ebXML[3]. In both cases, there are long-running conversations in which messages are exchanged, etc. However, the sequencing rules in tpaML and ebXML are significantly less powerful than provided by the state-machines in CPs. There is no way in tpaML or ebXML to do context-dependent

sequencing, or composition of sequencing rules, both of which are provided by nested CPs.

tpaML and ebXML are targeted at situations where two e-businesses, having already decided to do business together, want to negotiate agreements that completely specify their interoperability technology, frequently prior to any actual use of that technology. Thus there is little to support the needs of dynamic, flexible, evolving interaction patterns.

5.3 Web Services

The conversational model described above clearly differs in its crucial aspects from Web Services[4] as they are commonly understood today. We believe that with a few significant enhancements, however, the basic Web Services architecture can be extended to support the required features of the conversational model.

For example, message delivery would use the existing transport mechanisms as specified in WSDL. Businesses would use WSDL port types for endpoints that set up conversations and receive messages. UDDI entries would be used in the same way they are now, with the additional feature that businesses would list the top-level conversation policies they support.

The properties and requirements of conversation-enabled Web Services are explored in greater detail in ref. [14].

6 Ongoing and Future Work

We are currently working on two important aspects of a B2B integration architecture based on our conversational model: (1) Extension of the Java Connector Architecture (JCA) to support conversations; and (2) Definition of an XML-based language for scripting conversation policies. These are briefly discussed below.

6.1 Conversational JCA

The JCA architecture[1] provides a set of abstractions for connecting the J2EE platform to heterogeneous Enterprise Information Systems (EISs). The abstractions, defined as a set of contracts at the system and at the application level provide a collection of scalable, secure and transactional mechanisms that enable the integration of EISs with application servers and enterprise applications.

In this work we propose to extend the JCA application and system contracts to support conversations and conversation policies. This will extend the architecture from the realm of EIS integration to cross-enterprise integration. The (conversational) adapters built conforming to the architecture, provides the guarantee of being able to run on any J2EE platform and avail of the system specific resources (transaction, security, connection pooling and conversation management).

Conversational JCA are currently under development and are planned for public release on the IBM alphaWorks site.[15]

6.2 Conversation Policy XML

Business process integration through conversational interactions can only succeed on a wide scale if some common way of specifying CPs, is adopted as an industry standard.

For this reason, we are currently developing Conversation Policy XML (cpXML), an XML dialect for describing CPs. It permits CPs to be downloaded from third parties (such as standards bodies, providers of conversation-management systems, or even specialized protocol-development shops). Once downloaded and installed in a firm's conversation-management system, bindings are added to specify the connections between the decision points of the CP and the firm's business logic.

cpXML is narrowly scoped, restricting itself to describing the message interchanges exactly as we sketched them in Section 2. Thus, for example, it does not cover the way in which a CP is bound to the business logic, or the means by which the CP connects to the messaging system. It takes a third-party perspective, describing the message exchanges in terms of "roles" which are assumed at runtime by the businesses engaged in a conversation. It supports nesting of conversation policies, and time-based transitions such as timeouts on waiting for an incoming message.

Designing protocols is a daunting task, especially in B2B interactions, where the participants and the business needs are constantly changing, where the goals of the different parties are often at odds, and where there is no central authority to enforce conformance to any one protocol. The narrow scoping of cpXML significantly lightens the task of developing useful interaction patterns into formal, executable conversation policies. And while a state-machine-based model can be limiting in certain contexts, this is more than made up for by the clarity, simplicity, and ease of implementation that a state-machine affords.

7 Conclusion

We believe that conversation support technologies will greatly enhance the speed of e-business integration. Modeling of complex B2B interactions, which themselves follow a conversational pattern, fits naturally with

conversation policies. Extending standard integration architectures like JCA to include conversation support will allow tool vendors and application server providers to build applications that extend the realm of integration from EIS to cross-enterprise integration.

8 References

[1] J2EE Connector Architecture 1.0 Specifications from http://java.sun.com/j2ee/connector/

[2] Electronic Trading Partner Agreement for E-commerce (tpaML) draft version 1.0.6, from http://ebxml.org/project_teams/trade_partner/tpaml106.zip

[3] ebXML, http://ebxml.org

[4] Web Services, http://alphaworks.ibm.com/webservices

[5] Peregrine B2B Integration Platform, www.peregrine.com

[6] Thatte, S., "XLANG: Web Services for Business Process Design", Microsoft Corp., 2001. cf. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/

[7] WebMethods B2Bi www.webmethods.com.

[8] Vitria Business Ware. www.vitria.com.

[9] TIBCO Active Exchange. www.tibco.com.

[10] BPMI.org, "Business Process Modeling Language", http://www.bpmi.org/bpml-spec.esp

[11] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker, "Managing Process and Service Fusion in Virtual Enterprises", Information Systems, Vol. 24, No. 6, 1999, 429-456.

[12] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns – Elements of Reusable Object Oriented Software," Addison-Wesley Publishing Company, NY, 1995.

[13] Kumaran, S., Huang, Y., Chung, J-Y. A Framework-based Approach to Building Private Trading Exchanges. IBM Systems Journal (To Appear in June 2002).

[14] J. Hanson, P. Nandi and D. Levine, "Conversation-enabled Web Services for Agents and E-business", Proceedings of the Third International Conference on Internet Computing (IC-2002), to appear.

[15] http://www.alphaworks.ibm.com