Business Process Choreography for B2B Collaboration

This article describes a proposed methodology for business process choreography. It focuses on two types of business processes (contract and executable) and provides an interface protocol to represent interoperability patterns between them. The approach is designed to let existing processes, usually managed by an enterprise’s own internal workflow management system, collaborate.

Providing services cost-effectively and rapidly — especially in today’s challenging e-business environment — forces companies to interchange documents and information with many different business partners. In such an environment, difficult-to-manage business processes inevitably become more entangled and require collaboration between increasingly distributed and heterogeneous platforms.

The Gartner Group defines business process management (BPM) as “a set of services and tools that provide explicit process management (for example, process analysis, definition, execution, monitoring, and administration), including support for human and application-level integration.”¹ BPM touches on workflow, enterprise application integration, business-to-business (B2B) integration, and business process reengineering, automation, and integration.²

In an effort to realize BPM, researchers have proposed several specifications for conceptual models and structural syntax that also allow the incorporation of external Web services.³ Unfortunately, such proposals do not yet solve the problem of combining existing workflow processes seamlessly into collaboration design. Although several researchers have studied the issue of business process integration,⁴ their work is limited to B2B collaboration. More practical e-business environments, such as supply chain management, require a more systematic collaboration at the process, service, and application levels.

In this article, we propose a methodology for business process choreography, which we define as a procedure that incorporates existing workflow into business logic and generates a collaborative business process. Our methodology suggests a two-process specification along with a protocol specification to represent interactions between processes. We also describe the prototype cho-
Collaborative Business Processes

Essentially, our business process choreography is a formal methodology for representing interoperability patterns between two business processes and for automating the patterns systematically.

In our approach, we characterize a collaborative business process as a particular contract among business partners. This contract should clearly describe how to potentially associate several of the partners’ internal processes. Because the contract itself has a logical procedure among internal processes, we can represent it as a form of business process. We call the logical procedures among internal processes contract processes (CPs) and the partners’ own internal processes executable processes (EPs). Our interface protocol specifies any interactions between CPs and EPs.

To clarify,

- CPs define the procedural business transactions each partner participates in or performs for collaboration purposes. A CP is a sequence of business logic containing elements of data formats, logical end points, security levels, and so on. We can express CPs with recently proposed specifications such as the Business Process Modeling Language (BPML), the Business Process Execution Language for Web Services (BPEL4WS), and ebXML’s Business Process Specification Schemas (BPSS).
- EPs represent the internal, routine processes performed by the individual business partners involved in a CP. Usually, the partners’ own workflow management system (WFMS) controls the EP, but we can specify it by using the XML Processing Description Language (XPDL), which is a standard workflow definition language. An EP itself might not have any relationship with a specific CP, but it can be coupled to it via the interface protocol.

- The interface protocol describes the interoperability relationships that one or more of a business partner’s EPs have with a shared CP. These relationships are expressed via interoperability patterns, which we discuss later.

Figure 1 shows the relationship among CPs, EPs, and the interface protocol in a Web-based B2B environment. The figure illustrates three possible scenarios for an organization interacting with its partners through our business process choreography approach. In the first scenario, the organization exploits only Partner 1’s external application services (for example, Web services) through the CP. In the second, the organization’s CP interacts with Partner 2’s EP as well as Partner 1’s external application services. The last scenario shows an independent collaboration, in which the organization’s CP interacts with Partner 3’s CP, as well as its EP and application services.

Our approach has several advantages over other methods:

- **Reusability.** Because our choreography approach does not require any EP modification or adaptation, the EPs are totally reusable. Rather than using different workflow process definitions for different partners, organizations can use common definitions for identical internal processes.
- **Independence.** The interface protocol guarantees design independence because you can design an EP or CP without considering how to incorporate them with the protocol. Moreover, if you must modify EP or CP business logic, you can modify the CP or EP independently (if the modified parts aren’t related to the protocol).
- **Flexibility.** As Figure 1’s third scenario illustrates, one CP can interact simultaneously with external application services or other EPs or CPs. An organization can collaborate flexibly with its partners according to the required integration level.
We can design a collaborative business process by using mutually independent EPs and CPs, connected with interface protocols.

**Process Interoperability**

In a B2B collaboration environment, various patterns of interaction exist among the different business processes. To control this interaction effectively, we must first identify and then formally represent interoperability patterns.

**Interoperability Patterns**

We analyzed various types of interoperation between business processes and identified six primitive interoperability patterns as building blocks for expressing complex interactions. These primitives extend from the Workflow Management Coalition’s (WfMC’s) three interoperability models: chained, nested, and synchronized.⁶

In a chained model, one process triggers another process’s creation and enactment, but it takes no further interest in what happens next. This model subdivides into two types of patterns:

- In a *chained substitutive* (CS) pattern, the process terminates immediately after initiating the new process. As a result, the invoked process replaces the terminated process’s subsequent activities.
- In a *chained additive* (CA) pattern, the process follows its own execution after initiating the new process. The two processes are enacted in parallel, but they don’t interact with each other again.

In a nested model, the invoking process takes execution results from the invoked process at a particular activity. We can subdivide this model into three patterns:

- In the *nested synchronous* (NS) pattern, the invoking process is blocked until it takes back the execution results at the same activity at which it invokes the other process. The invoked process plays the role of a subprocess specializing in an activity within the invoking process.
- In the *nested deferred* (ND) pattern, a return point defers to a certain activity that comes after the invocation point. The new process invalidates or overrides the intervening activities between the two points.
- The *nested parallel* (NP) pattern is the same as the ND pattern, except that all the intervening activities are processed in parallel with the new process, instead of invalidated.

The synchronized model follows only one pattern — *parallel synchronized* (PS) — in which two processes synchronize at a specific point. Only after both of them reach that point can they continue their execution.

Figure 2 shows the primitive interoperability patterns that can occur between two processes. In chained models, activity c and subsequent activities in the CS pattern abort, while those in the CA pattern continue. In nested models, invoking processes in the NS and ND patterns are suspended after activity a invokes activity b, whereas the invoking process in the NP pattern continues and activates activity c. In the synchronized model, activities c and d can be activated only after both activities a and b complete.

**Expression of Interoperability Patterns**

To formally specify interoperability patterns, we define five operations required for a process to initiate or activate another process’s services. In particular, these operations facilitate message
exchanges or event notifications among business processes.

First, a process instance must make a connection with another process instance. Instantiate operations request another party’s enactment system to create an instance of a target process and return that instance’s key. Initiate operations ask the system to find one of the target process’s existing instances waiting to be invoked after its previous activities are done.

Next, an invoked process instance might need operations to reinteract with the process instance that invoked itself with Instantiate or Initiate operations. The invoked instance’s Resume operation instructs the invoking instance, which is waiting or has been suspended after invoking, to resume its target activity immediately. But the invoked instance’s Transit operation only allows a continuation of the invoking instance, which can start the target activity immediately after its previous activities are completed. Finally, Synchronize operations express synchronized patterns — that is, the operations make two process instances continue their next activities only after their appointed activities are done.

Table 1 compares interoperability operation characteristics. The first two columns show that Transit, Resume, and Synchronize operations should follow Instantiate and Initiate operations because the former set can interact with process instances the latter set appoints. The next two columns instruct the enactment systems on how to behave according to each operation. The enactment system simply needs to comply with the request immediately if it receives Instantiate or Resume operations. Initiate, Transit, and Synchronize operations check the system to ensure the invoked activity’s preceding activities have been completed in the process instance.

All operations except Synchronize have state attributes that describe the invoking processes’ states after sending the operations. This state attribute can have one of five values:

- **Waited** state means that the invoking activity in an invoking process will wait for the invoked process’s reply.
- **Suspended** state means that an invoking process will be suspended until it receives a reply from the invoked process (but the resumed activity is not the same as the invoking activity).
- **Terminated** state means that an invoking process will be terminated and won’t continue any subsequent activity.
- **Disconnected** state means that an invoking process will continue, but it will have no more interactions after sending the operation.
- **Continued** state means that an invoking process will continue its subsequent procedure by the appointed activity in which the process will receive a reply of the invoked process.

To implement these operations effectively, the Wf-XML standard provides specifications that facilitate XML-based communication between heterogeneous workflow engines. For instance, the Initiate operation can request another process engine to create a target process instance by sending the CreateProcessInstance message, as defined by the Wf-XML specification.

By composing these operations, we can express Figure 2’s primitive interoperability patterns. Table 2 shows the patterns’ expressions; for example, using an Instantiate or Initiate operation expresses chained models. To create a new invoked process instance, we use an Instantiate operation; to select it from among instances waiting in the appointed activity, we use an Initiate operation. Both operations can have terminated or disconnected states, which means the interoperation follows a CS or CA pattern, respectively. In the same way, we can express the NS pattern with Instantiate (state=‘waited’) and Resumed(state=any), or Initiate(state=‘waited’) and Resumed (state=any). The starting operation’s waited

<table>
<thead>
<tr>
<th>Operations</th>
<th>Execution order</th>
<th>Execution type</th>
<th>State attribute</th>
<th>WF-XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instantiate</td>
<td>Connector</td>
<td>Immediate execution</td>
<td>Yes</td>
<td>CreateProcessInstance</td>
</tr>
<tr>
<td>Initiate</td>
<td>Connector</td>
<td>Check transition</td>
<td>Yes</td>
<td>Notify</td>
</tr>
<tr>
<td>Transit</td>
<td>Follower</td>
<td>Check transition</td>
<td>Yes</td>
<td>Notify</td>
</tr>
<tr>
<td>Resume</td>
<td>Follower</td>
<td>Immediate execution</td>
<td>Yes</td>
<td>ChangeProcessInstanceState</td>
</tr>
<tr>
<td>Synchronize</td>
<td>Follower</td>
<td>Check transition</td>
<td>No</td>
<td>Notify</td>
</tr>
</tbody>
</table>

Business Processes on the Web
state means the invoking process starts waiting for the invoked process’s reply in the invoking activity itself. In the ND and NP patterns, the starting operations have suspended and continued states, respectively. Subsequent operations, such as the Resume or Transit operations in nested models, can have any state value according to potential behaviors. The subsequent operations’ states are especially useful in expressing hybrid interoperability patterns. Finally, we can express the PS pattern using a Synchronized operation with no state.

We can extend primitive interoperability patterns into hybrid patterns by combining them. Figure 3 illustrates hybrid patterns that combine two arbitrary primitive patterns. We can’t blend chained patterns with other patterns because we assume they aren’t connected after invoking a new process.

We also can express hybrid patterns with interoperability operations. For example, in Figure 3, we can represent NS®ND using the three operations of Instantiate(state=’waited’), Resume(state=’suspended’), and Resume(state=’continued’). The same is true for the other hybrid patterns.

**Business Process Choreography**

Four steps comprise an overall procedure for business process choreography. First, all participants make interoperability contracts and extract business logic, and then they design a common CP for a collaborative business process. Second, participants each check their own internal processes and prepare the EPs necessary for the business logic. Then, the participants analyze the relationships between the common CP and their own EPs. Finally, each defines an interface protocol, which formally specifies the interactions between the CP and the participant’s own EPs.

Figure 4 (next page) illustrates a purchasing process between a customer and a supplier. The CP in the figure defines the business logic and message exchanges the participants must perform in the purchasing process. The four EPs in the figure show the customer and supplier workflow processes. RequestOrder EPc and CheckInvoice EPc are the customer’s own internal processes for Purchasing CPc. The supplier also participates in the purchasing process with CheckOrder EPo and CreateInvoice EPo.

To put the interactions into operation, the customer defines an interface protocol by associating RequestOrder EPc and CheckInvoice EPc with Purchase CPc. We assume EPc is encapsulated and EPc is not; we define the supplier’s interface protocol in the same way.

Now, we translate every interaction in the interface protocol into an interoperability pattern and then specify the pattern via interoperability operations. For instance, because the interoperability pattern between EPc and CPc corresponds to the NP pattern, the Instantiate(state=’continued’) operation expresses the starting interaction between EPc and CPc; the Transit(state=’continued’) and Transit(state=’terminated’) operations, which are from sendAcceptance and sendRejection activities, respectively, describe the two subse-
sequent interactions. We cannot associate CP\textsubscript{P} directly with EP\textsubscript{C} activities because EP\textsubscript{C} is encapsulated. Instead, we use the EP\textsubscript{C} for instantiating its termination and notifying the invoking process CP\textsubscript{P}. The interoperability pattern between the two processes corresponds to the NS pattern, and we can express EP\textsubscript{C} with the \texttt{Instantiate(state = 'waited')} and \texttt{Transit(state = 'terminated')} operations.

Figure 5 shows how purchasing processes interact between the customer’s common CP and EPs. The interface protocol contains information on the interoperability patterns and the message transformation. All operations in the patterns can be transformed to corresponding Wf-XML messages with input/output parameters. For example, if activity sendPO in EP\textsubscript{R} sends a CreateProcessInstance.request message with ObserverKey, ContextData, and so on, then IP transforms purchaseOrder to PO schema and requests CP\textsubscript{P} to instantiate. When the new CP\textsubscript{P} instance key returns to EP\textsubscript{R} via CreateProcessInstance.response with ProcessInstanceKey, the first interaction is completed.

We can describe Figure 5’s Purchase IP\textsubscript{P} in XML as follows:

```xml
<InterfaceProcess Id="IP_P" Name="Purchase IP" xmlns="...">
  <ContractProcess Id="CP_P" Name="Purchase CP" Encapsulated="No" Key="http://..."/>
  <ExecutableProcess Id="EP_R" Name="RequestOrder" Encapsulated="No" Key="http://..."/>
  <ExecutableProcess Id="EP_C" Name="CheckInvoice" Encapsulated="Yes" Key="http://..."/>
  <Coupling Id="1">
    <Instantiate From="EP_R" To="CP_P" State="continued">
      <Source Activity="epr:sendPO"
      ...
    </Instantiate>
  </Coupling>
</InterfaceProcess>
```
We base interface protocol specification on the XML schema we defined for the prototype system. The interface protocol specification for IP includes two coupling elements that join CP with the customer’s EP and EP, respectively. The first coupling element expresses the NP interoperability pattern, and the second does the NS pattern.

**System Design**

Figure 6 (next page) shows our prototype system’s overall architecture, including its two subsystems, WFMS and business process management system (BPMS). WFMS has its own storage, client tools, and engine (called WF_engine), and controls the EP. BPMS has storage and an engine (called BP_engine) for managing CP. It also has storage and an IP interpreter for processing the interface protocol.

Our system uses XML-based process definitions, which are stored in XML databases. Workflow process definitions are stored in EP storage, based on WfMC-defined XPDL specifications. Collabo-
ration process definitions are stored in CP storage, based on BPMI-defined BPML specifications.9 Finally, interface protocol storage stores IP specifications, based on the XML schema we’ve defined.

WF_engine and BP_engine manipulate process definitions via two techniques: XPath and the Java Architecture for XML Binding (JAXB). The engines communicate with each other through the interface protocol interpreter, which also helps CP and EP interact by interpreting interoperability pattern information, operations, and schema transformations in the interface protocol specification.

Each subsystem has a Wf-XML interpreter that translates messages and interoperability operations and an application adapter that supports automated tasks in workflow or business collaborations. Figure 7 shows our system modules’ operation sequence when the system executes part of a purchasing example.

**Future Work**

Our research lets companies in complex business environments reuse internal workflows effectively and control business interactions automatically. Malone and colleagues analyzed three types of dependencies among multiple activities according to the use of common resources.10 When considering such dependencies, our interoperability patterns can be supplemented or subdivided into various patterns. This issue of concurrent control of shared resources among two or more organizations is an interesting area for future research. In addition, if two partners have different schemas, the automated control of process interoperation requires data transformation.11 Data manipulation such as schema translation or common virtual schema generation is another interesting issue for future research in business process integration.

Several issues require further work. One is controlling the right of participants’ information access. Again, blocking the details of a business process from unauthorized participants is key. Another issue is process encapsulation.12 Encapsulating a business process conceals a detailed specification of it from external entities. In other words, if a process is not encapsulated, we can get information about how to invoke activities within it and then directly activate them. To interoperate with an encapsulated process, we must send appointed messages or raise particular events to a target enactment system that controls the process.

**References**

Choreography for B2B Collaboration

Figure 7. Business process choreography system modules’ operation sequence. Step one, the WF_engine sends a Wf-XML message with a purchase order to the interface protocol interpreter to start a new contract process (CP). Two, the interpreter executes the corresponding operation to have BP_engine initiate the contract process. Three, BP_engine creates and enacts the CP instance. Four, the interface protocol interpreter translates operations to Wf-XML messages and sends them to a predefined external WF_engine. Five, this WF_engine parses the Wf-XML messages, creates the workflow instance, and returns its key. Six, the WF_engine enacts the workflow instance. Finally, the internal WF_engine receives the Wf-XML messages and continues to enact the corresponding workflow process instance.

Jae-yoon Jung is a PhD candidate at Seoul National University. His research interests include workflow management systems, business process management, and B2B integration. He received his BS and MS in industrial engineering from Seoul National University. Contact him at jyjung@ara.snu.ac.kr.

Wonchang Hur is a PhD candidate at Seoul National University. His research interests include workflow management systems, the Semantic Web and its applications, artificial intelligence, and B2B e-commerce. He received his BS and MS in industrial engineering from Seoul National University. Contact him at bethoven@ara.snu.ac.kr.

Suk-Ho Kang is a professor of industrial engineering at Seoul National University. His research interests are in intelligent manufacturing systems and e-business. He received a BS in physics from Seoul National University, an MS from the University of Washington, and a PhD from Texas A&M University, both in industrial engineering. Contact him at shkang@cybernet.snu.ac.kr.

Hoontae Kim is an associate professor in the Department of Industrial and System Engineering at Daejin University. His research interests include process analysis and integration, and supply chain management. He received a PhD in industrial engineering from Seoul National University. Contact him at hoontae@daejin.ac.kr.