Business Process Choreography for B2B Collaboration

Jae-yoon Jung, Wonchang Hur, Hoontae Kim, and Suk-Ho Kang

Department of Industrial Engineering, Seoul National University, Seoul, 151-742, Republic of Korea

Abstracts

We propose a methodology for business process choreography. Our methodology provides specifications of two types of business processes (Contract Process and Executable Process), and an interface protocol to represent interoperability patterns between the processes. Through our approach, existing processes usually managed by companies' own internal WFMS can be put together to cooperate and controlled following a consistent procedure. We also implemented a prototype business process management system to support our business process choreography methodology. The system is built on top of existing WFMS's. It facilitates creation and instantiation of Contract Processes, and manages an automatic execution of the Interface Protocol.

Keywords: Business process choreography, Process interoperability patterns, B2B collaboration

1. INTRODUCTION

Today's e-business environment urges many companies to cooperate with each other. For cost-effective and rapid provision of good services, a company needs to interchange documents and related information with many business partners which include suppliers, customers, and various service providers. In such an environment, business processes inevitably get more entangled and entails collaboration between distributed and heterogeneous platforms which are not easy to manage. Therefore, a systematic and automated management of business process execution has drawn a great concern among companies and organizations that necessitate collaborative business process.

BPM (Business Process Management) can be defined as "a set of services and tools that provide explicit process management (e.g., process analysis, definition, execution, monitoring and administration), including support for human and application-level integration".[1] It is associated with a number of technologies, such as workflow, Enterprise Application Integration (EAI), B2Bi, and also with concepts such as Business Process Reengineering (BPR), Business Process Automation (BPA) and Business Process Integration (BPI).[2]

In an effort to realize the concept of BPM, various business process specifications are recently proposed, and they make it possible to define structure of a business process, exchangeable messages and operations. They also make it possible to incorporate external web services for modeling collaboration with external service providers.[3] However, they do not provide solutions to combine existing workflow or internal processes seamlessly in the collaboration design. In more intimate e-business environments such as supply chain management, a more active collaboration is needed at the process level and at the service and application level.

Published in IEEE Internet Computing, Vol. 8, No. 1, Jan/Feb 2004, pp. 37-45.

In this paper, we propose a business process choreography methodology. Our methodology provides two specifications that can be used to represent two types of business processes (Contract Process and Executable Process) and a protocol specification (Interface Protocol) used to represent interaction between the processes. Through our approach, existing processes which are usually managed by companies' own internal WFMS can be put together and controlled following a consistent procedure. We also implemented a prototype business process management system to support our business process choreography methodology. The system is built on top of existing WFMS's which manage Executable Processes, and facilitates creation and instantiation of Contract Processes, and management of automatic execution procedure of the Interface Protocol.

2. COLLABORATIVE BUSINESS PROCESSES

In this chapter, we first provide classification of business processes in a web-based B2B environment, and then describe how to organize them into collaboration. The essence of our business process choreography is a formal methodology to represent interoperability patterns between two business processes and to provide a way to systematically automate the patterns.

In our approach, we characterize a collaborative business process as a particular contract among business partners. The contract may involve several internal processes of the partners, and clearly describe how to associate the involved processes. Because the contract itself has a logical procedure, it also can be represented as a form of business process. We call the logical procedure of the contract *Contract Process (CP)*, and the partners' own internal processes *Executable Process (EP)*. A particular communication protocol, called *Interface Protocol (IP)*, is defined to specify interactions between CP and EP. These concepts can be defined more specifically as follows.

[Definition of Contract Process (CP)] CP defines procedural business transactions that each business partner participates in and carries out for the purpose of collaboration. It is described as a sequence of business logics that contain elements of data formats, logical endpoints, security levels, etc. CP can be expressed by using recently proposed specifications such as BPML, BPEL4WS and ebXML BPSS.

[Definition of Executable Process (EP)] EP represents an internal, routine process performed by individual business partners involved in CP. Usually, EPs are controlled by the partners' own WFMS, so it can be specified using XPDL, which is a standard workflow definition language. EP itself may not have any relation with a specific CP, but it can be related to the CP through IP.

[Definition of Interface Protocol (IP)] IP is used to describe interoperability relationships that one or more EP in a business partner interacts with one CP. The relationships are expressed by means of interoperability patterns which will be discussed in the next chapter.

Figure 1 shows the relationship among CP, EP and IP in a web-based B2B environment. The figure illustrates three possible scenarios of an organization interacting with other partners through our business process choreography concept. In the first scenario, the organization communicates only with external application services (e.g., web services) of the *Partner 1* through CP. In the second, the organization's CP interacts with EP of the *Partner 2* as well as external application services of the *Partner 1*. The last one shows an independent collaboration scenario, in which the organization's CP interacts with CP of all the partners, as well as EP and web service.

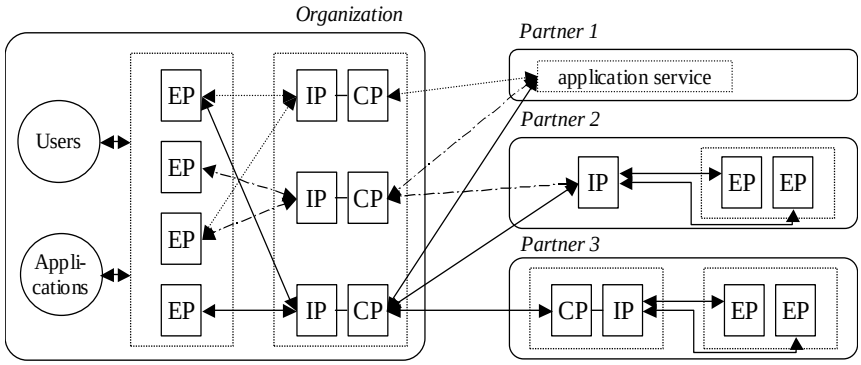


Figure 1. Collaboration scenarios

This approach has several advantages as follows over other approaches.

- *Reusability* – Our choreography methodology does not require any modification or adaptation of EPs, therefore EPs are totally reusable. We do not need to use different workflow processes definitions for each partner, but use common definitions if they are identical. Reuse of EP helps agile and flexible response to new business requirements.
- *Independence* – IP guarantees design independence of EP and CP. In other words, design of IP has no effect on that of EP or CP, which implies EP and CP can be designed without considering how it will be incorporated in IP. Moreover, if it is necessary to modify some business logics of EP or CP, we can modify the CP or EP independently, if the modified parts are not related to IP.
- *Flexibility* – An organization only has to modify IP, not CP or EP, if he wants to make a business transaction with a new partner. The process logics of the organization need not to be modified or altered. Because CP supports three collaboration scenarios, the organization can flexibly collaborate with new business partners in various types of collaboration environments, shown as Figure 1.

3. PROCESS INTEROPERABILITY

In a B2B collaboration environment, there can be various patterns of interaction among business processes. To support an effective control of the interaction, it is required to identify interoperability patterns and formally represent the patterns. In this chapter, we provide a systematic methodology to facilitate the formal representation of the interoperability patterns.

3.1 Interoperability Patterns

We analyzed various types of interoperation between business processes, and identified 6 primitive interoperability patterns which can be used as building blocks to express complex interactions. These primitives are extended from WfMC's interoperability models: chained model, nested model, and synchronized model.[4]

First of all, in a chained model, a process triggers the creation and enactment of another process, and then takes no further interest in the newly created process. This model is subdivided into the following two types.

- *Chained Substitutive (CS)* – a process terminates right after initiating a new process. The newly created process replaces the original process.
- *Chained Additive (CA)* – a process just goes on its own execution after initiating a new

process. Two processes do not interact again with each other.

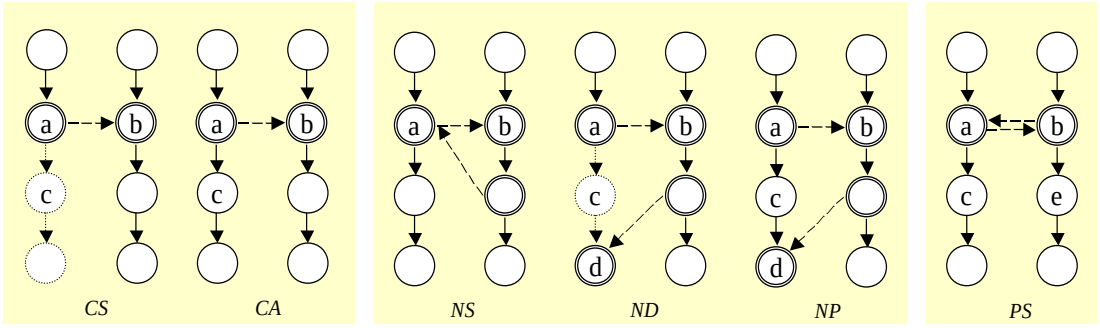
Secondly, in a nested model, after a process invokes another process, the invoking process takes execution results from the invoked process at a particular activity. We subdivide this model into three patterns.

- *Nested Synchronous (NS)* – a return point where a process takes back the execution results is the same as the invoking point of the process. The invoked process plays a role of a sub-process substituting an activity in the invoking process.
- *Nested Deferred (ND)* – a return point is deferred to a certain activity that comes after the invoking point. The intervening activities between the two points are overridden by the new process.
- *Nested Parallel (NP) – NP* is the same as *NR* except that the invoking process can be activated after all the intervening activities are completed.

Finally, the synchronized model is described as follows.

- *Parallel Synchronized (PS)* – two processes are synchronized at a specific point. Only after both of them reach the point, they can continue their execution.

Figure 1 shows the primitive interoperability patterns which can happen between two processes.



(a) chained (b) nested (c) synchronized

Figure 2. Primitive interoperability patterns between two processes

One thing that you need to consider here is the concept of process encapsulation.[5] Encapsulating a business process is to conceal a detailed specification of the process from external entities. In other words, if a process is not encapsulated, we can get information about how to invoke activities in the process, and directly activate them. To interoperate with an encapsulated process, we have to send messages or raise events to a target enactment system that controls the process.

3.2 Expression of Interoperability Patterns

To formally specify the interoperability patterns, we define 5 interoperability operations. The operations represent atomic functions that are required for a process to initiate or activate services of another process. The operations facilitate messages exchange or event notification between business processes.

First of all, it is required that a process instance makes a connection with a certain process instance. *Instantiate* operation requests the other party's enactment system to create an instance

of a target process and return the key of the instance. *Initiate* operation requests the system to find one of existing instances which wait for a connection after previous activities have been done.

Next, an invoking process instance needs operations to interact with the process instance which has been decided by *Instantiate* or *Initiate* operation. *Resume* operation notifies the invoking instance, which are waiting or suspended immediately after its invocation, to continue its next activities. But, *Transit* operation sends the notification of continuation to the invoking instance which has done or been doing appointed activities after its invocation. Finally, *Synchronize* operation expresses Synchronized pattern, that is, the operation makes two process instances continue their next activities after both of their appointed activities are done.

The characteristics of interoperability operations are compared in Table 1. First two columns shows *Transit*, *Resume* and *Synchronized* operations should follow *Instantiate* and *Initiate* operations. And in the next two columns, Instantiate and Resume operations have the enactment system treat the request immediately. But *Initiate*, *Transit*, *Synchronize* operations have the enactment system check transitions of the target activity because the system should examine whether or not the previous activities of the instance have been done.

Table 1. Process interoperability operations

operations	connector	follower	immediate execution	check transition	state attribute	Wf-XML
Instantiate	O		O		O	*CreateProcessInstance*
Initiate	O			O	O	*Notify*
Transit		O		O	O	*Notify*
Resume		O	O		O	*ChangeProcessInstanceState*
Synchronize		O		O		*Notify*

Additionally, all the operations, except *Synchronize*, have 'state' attributes which describe states of invoking processes after the execution of operations. The 'state' attribute can have one of the following values: *waited*, *suspended*, *terminated*, *disconnected*, and *continued*. And, *Synchronize* operation implicitly has "*continued*" value as its 'state' attribute. The state of *Initiate* and *Invoke* operations decides interoperability patterns of two processes.

To support the effective implementation of the operations, Wf-XML messages, shown in the table, may be exploited. The Wf-XML standard provides XML specifications that facilitate XML-based communication between heterogeneous workflow systems.[6] For instance, the *Initiate* operation can request another process engine to create a target process instance by sending the "*CreateProcessInstance*" message in the Wf-XML specification.

By composing theses operations, we can express the primitive interoperability patterns described in the previous section. The expressions of the patterns are shown in Table 2. For example, the pattern *NS* can be expressed by using two operations of *Instantiate(state='waited')* and *Resumed*, which implement a sub-process overriding an activity.

Table 2. Expression of primitive interoperability patterns by interoperability operations

Pattern		Expression
Chained	*CS*	*Instantiate/Initiate(state='terminated')*
	CA	*Instantiate/Initiate(state='disconnected')*
Nested	*NS*	*Instantiate/Initiate(state='waited')* → *Resume*

	ND	*Instantiate/Initiate(state='suspended')* → *Resume*
	NP	*Instantiate/Initiate(state='continued')* → *Transit/Synchronize*
Synchronized	*SP*	*Synchronize*

Noticeably, the primitive interoperability patterns can be extended to hybrid patterns by combining each other. Figure 5 illustrates the hybrid patterns combining two arbitrary primitive patterns except chained patterns. They cannot be blended with the other patterns because their connection is lost after a new process starts.

The hybrid patterns also can be expressed by the interoperability operations. For example, in the figure, *NS⊕ND* can be represented by using three operations of *Initiate(state= 'waited')*, *Resume(state='suspended')* and *Resume(state='continued')*. The other hybrid patterns can be expressed in the same way.

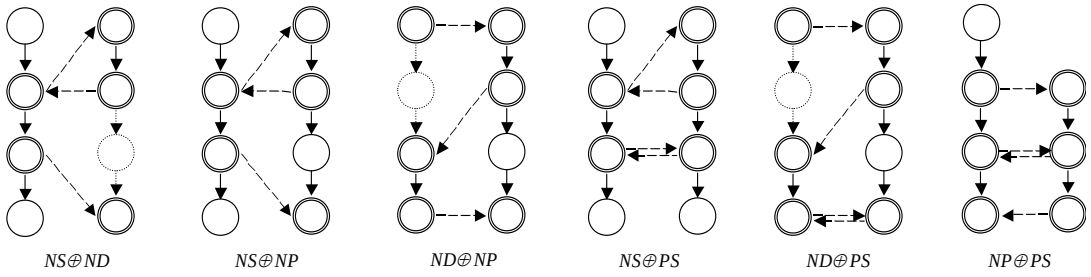


Figure 3. The hybrid interoperability patterns between two processes

4. BUSINESS PROCESS CHOREOGRAPHY

An overall procedure for business process choreography consists of four steps. Firstly, all participants make interoperability contracts and extract business logics together, and they design a common CP for a collaborative business process. Secondly, each participant checks out his own internal processes, and prepares EPs which are necessary for B2B collaboration. Finally, each participant analyzes relationships between the common CP and his EPs, and defines his own IP which formally specifies interactions between the CP and his EPs.

Figure 3 illustrates a CP for a purchasing process between a customer and a supplier. The CP defines business logics and message exchange for the participants to perform the purchasing process. And four EPs in the figure show workflow processes of a customer and a supplier. *RequestOrder* EP_R and *CheckInvoice* EP_C are the customer's own internal process for *Purchasing* CP_P. The supplier also participates in the CP with *CheckOrder* EP_O and *CreateInvoice* EP_I.

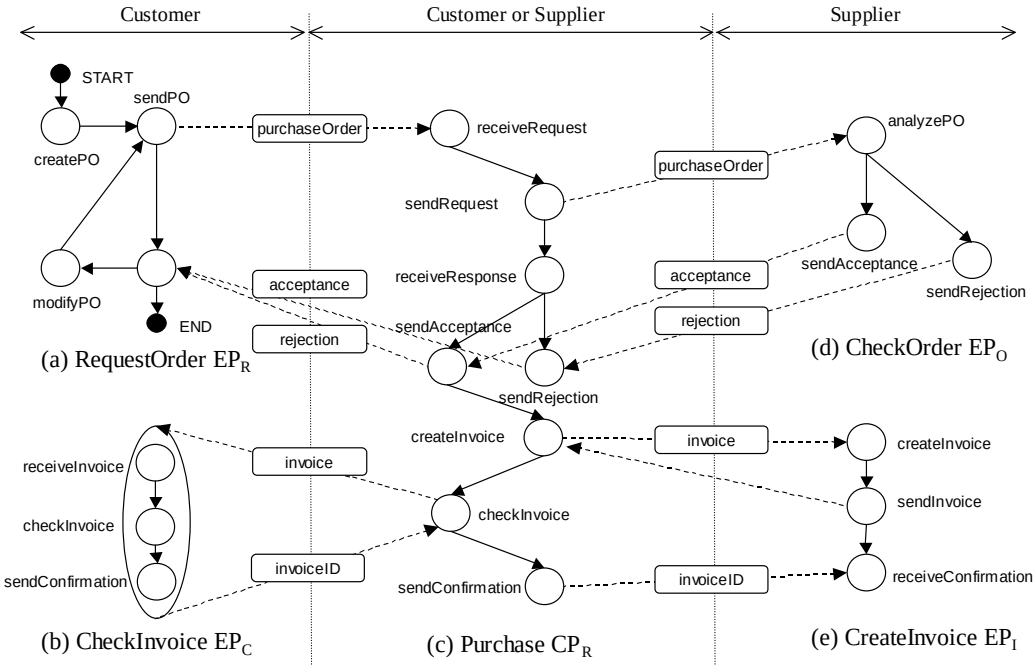


Figure 4. An example of a purchasing process

To put the interactions into operation, in the figure, the IP for the customer is defined by associating *RequestOrder* EP_R and *CheckInvoice* EP_C with *Purchase* CP_P. The IP for the supplier is also defined in the same way. We assume that EP_C is encapsulated and EP_R is not.

Now, every interaction in IP is translated into the interoperability patterns, and then specified by using the interoperability operations. For instances, as the interoperability pattern between EP_R and CP_P corresponds to the primitive pattern *NP* as described in the previous chapter, the first interaction between EP_R and CP_P is expressed by *Instantiate(state='continued')* operation and he second by *Transit(state= 'continued')* or *Transit(state= 'terminated')* operation. In the case of EP_C, we can not associate CP_P directly with activities of EP_C because of its encapsulation. Instead, the EP_C is used only for instantiation and notification of its termination to the invoking process CP_P. We can easily find that and the interoperability pattern between them corresponds to *NS*, and EP_C can be expressed by *Instantiate(state='waited')* and *Transit(state= 'terminated')* operations

Figure 4 shows how business processes for purchasing are interacted between common CP and EPs of the customer. IP contains information of the interoperability patterns and message transformation. And all operations in the patterns are matched to corresponding Wf-XML messages with input/output parameters. For example, if activity *sendPO* in EP_R sends "*CreateProcessInstance.request*" message with *ObserverKey*, *ContextData*, and etc., IP translates "*purchaseOrder*" to "*PO*" schema, and requests CP_R of its instantiation. At last, when the key of new CP_P instance is returned to EP_R by means of "*CreateProcessInstance.response*" with *ProcessInstanceKey*, the first interaction is completed.

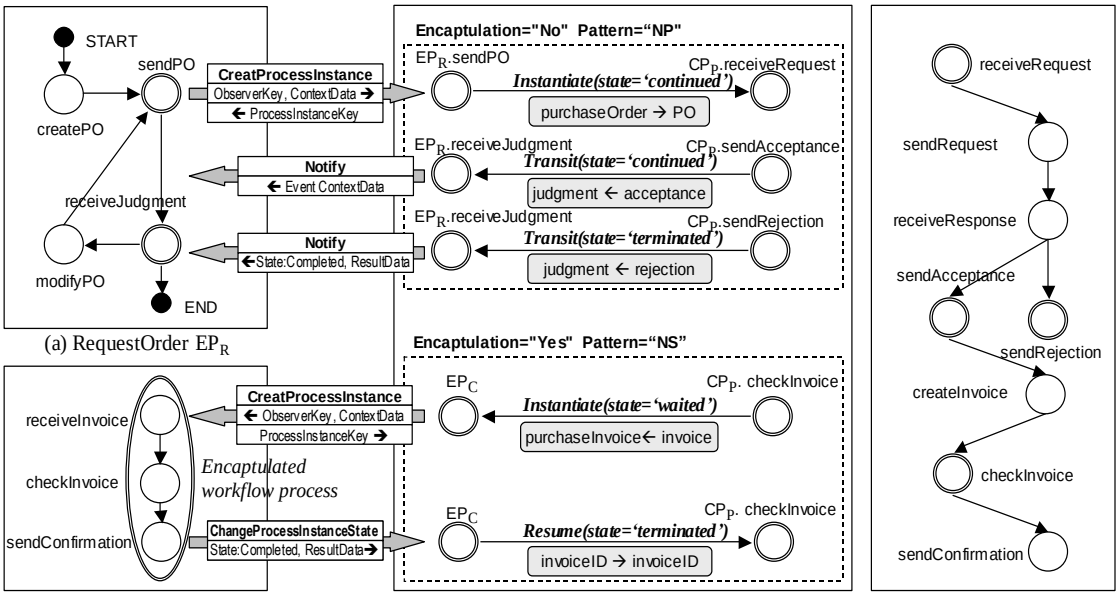


Figure 5. IP design for a purchasing process

The *Purchase IP*$_P$ in Figure 4 is described in XML specification as follows. The IP specification is based on XML Schema which we defined for the prototype system. It has two Coupling elements to couple *EP*$_R$ and *EP*$_C$ with *CP*$_P$. The first Coupling element expresses interoperability pattern *NP*, and the second does pattern NS.

```
<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" InputData="epr:purchaseOrder"/>
         <Destination Activity="cpp:receiveRequest" OutputData="cpp:PO"/>
      </Instantiate>
      <Transit From="CP_P" To="EP_R" State="continued">
         <Source Activity="cpp:sendAcceptance" InputData="cpp:acceptance"/>
         <Destination Activity="epr:receiveJudgment" OutputData="epr:judgment"/>
      </Transit>
      <Transit From="CP_P" To="EP_R" State="terminated">
         <Source Activity="cpp:sendRejection" InputData="cpp:rejection"/>
         <Destination Activity="epr:receiveJudgment" OutputData="epr:judgment"/>
      </Transit>
   </Coupling>

   <Coupling Id="2">
         …
   </Coupling>
</InterfaceProcess>
```

5. SYSTEM DESIGN

We implemented a prototype system to support business process choreography. The overall architecture of the system is presented in Figure 5. The system consists of two sub-systems of WFMS and BPMS. WFMS has its own storage, client tools and engine(WF_Engine), and takes the charge of EP control using them. BPMS has storage and an engine for managing CP(BP_Engine). It also has storage and an interpreter for processing IP(IP interpreter).

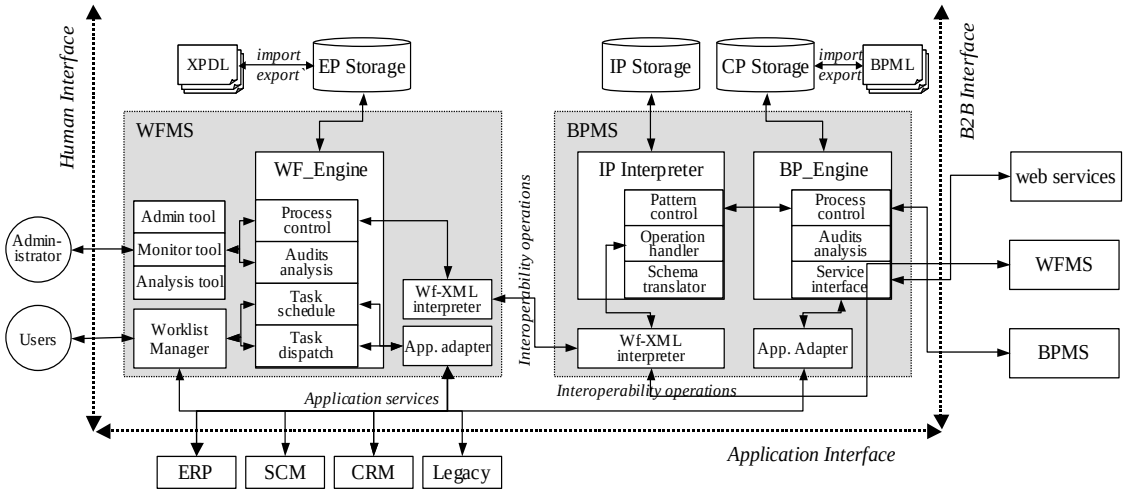


Figure 6. Overall architecture of our business process choreography system

The system uses XML-based process definitions, which are stored in XML database. In detail, workflow process definitions have been stored in EP storage, based on XPDL specifications, defined by WfMC.[7] And collaboration process definitions have been stored in CP storage, based on BPML specification, defined by BPMI.[8] Finally, IP Storage has stored IP specifications, which follows XML schema we defined.

WF_Engine and BP_Engine are enactment engines of WFMS and BPMS, respectively. Two engines manipulate the process definitions by two techniques, XPath and JAXB(Java Architecture for XML Binging). And, the engines communicate with each other through the IP interpreter. IP interpreter helps CP and EP to interact, based on information of interoperation patterns, operations and schema transformation in IP specification.

Both of two systems have Wf-XML interpreter and Application adapter. Wf-XML interpreter plays a role in translator between messages and interoperability operations. Application adapter supports automated tasks in workflow or business collaborations.

Figure 6 illustrates an operation sequence of our system following the execution scenario of a purchasing order process.

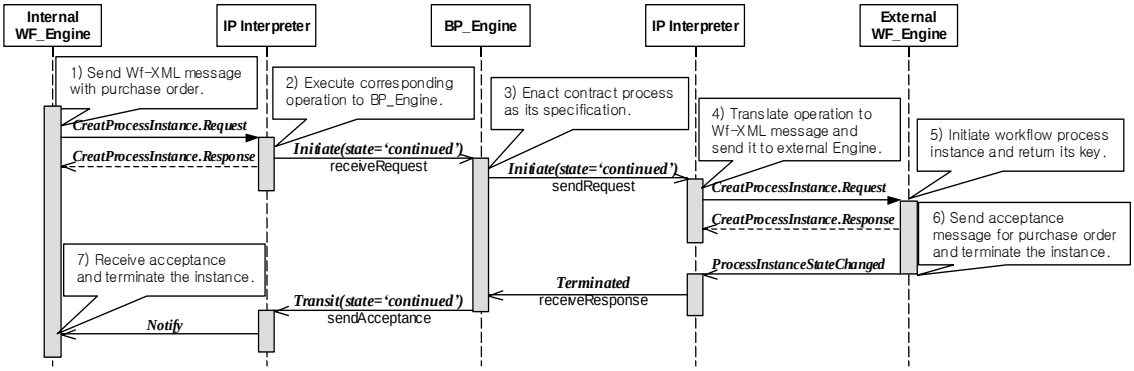


Figure 7. Overall architecture of our business process choreography system

1) Internal WF_Engine sends a Wf-XML message with purchase order to IP interpreter in order to start a new contract process.

2) IP interpreter execute corresponding operation to BP_Engine in order to initiates the contract process based on IP specifications.

3) BP_Engine creates and enact the contract process instance. And if it wants to interact with workflow, it requests for IP interpreter to perform operations in corresponding IP specification.

4) IP interpreter translates operations to Wf-XML messages and send them to pre-defined Wf_Engines, which can be external WF_Engines.

5) External WF_Engine parses the Wf-XML message. It creates the workflow process instance and returns its key.

6) The Wf_Engine enacts the workflow instance. If the instance transits or changes its state, the engine sends Wf-XML messages to its corresponding IP interpreter.

7) Internal WF_Engine receives the Wf-XML messages and transits the corresponding workflow process instance.

6. CONCLUSION

In this paper we proposed a methodology of choreographing business processes to achieve an automated control of collaborative interactions between business partners. The advantage of our approach is that we can establish flexible and extensible interactions between partners involved in B2B collaboration without modifying a structure of business processes of them. This is because we separated procedural business logics required to complete the interactions from the individual processes involved, and adopted a coupling technique based on interoperability patterns between business processes.

References

1. Gartner group, "Impact of BPM on Application Development," *Gartner Symposium 2001*, Orlando, Fla., 2001.

2. J. Pyke, "What's happened to workflow?" *Information Management & Technology*, vol. 35, no. 6, 2002, pp.254-256.

3. C. Peltz, "Web services orchestration- a review of emerging technologies, tools, and standards," *Hewlett Packard, Co*., 2003.

4. *WFMC-TC-1012, Workflow Standard—Interoperability Abstract Specification,* Workflow Management Coalition, Winchester, UK, 1999.

5. Y. Kim et al., "WW-Flow: Web-Based Workflow Management with Runtime Encapsulation," *IEEE Internet Computing*, vol. 4, no. 3, May/June 2000, pp.55-64.

6. *WFMC-TC-1023, Workflow Standard—Interoperability Wf-XML Binding,* Workflow Management Coalition, Lighthouse Point, Fla., 2001.

7. *WFMC-TC-1025, Workflow Process Definition Interface—XML Process Definition Language,* Workflow Management Coalition, Lighthouse Point, Fla., 2002.

8. A. Arkin, *BPMI Proposed Recommendation, Business Process Modeling Language,* Business Process Management Initiative, 2003.