An Agent-based Process Environment for Electronic Commerce

Jen-Yen Chen and Yung-Jui Chen

Dept. of Computer Science and Information Engineering
Hsinchu, Taiwan

Abstract

Electronic commerce process traditionally is enacted by centralized work flow management system which is inflexible and unreliable. Contrarily, the paper presents a decentralized architecture composed of collaborating agents to solve the problem. Also, mobile agent is used to reduce communication burden on the web.

Moreover, both process modeling language and agent communication message are modeled by extensible markup language (XML) which is regarded as the universal format on the web. This greatly improves interoperability which is very important, as electronic commerce process normally involves multiple systems with different process modeling languages.

This process environment is currently implemented in JAVA, and is based on the IBM aglets software development kit (ASDK).

Key Words: process environment, agent programming, XML (extended markup language)

1. Process Enactment Using Agents

Business process, especially electronic commerce process, is very dynamic and unpredictable, making it difficult to define its activities and work flow (process). While work flow management system (WFMS) [1,2,5,6] provides definition, generation, management, and enactment of work flow, WFMS can thus be used in electronic commerce (e-commerce) process. However, WFMS lacks the capabilities of evolution meaning that flow can be changed during enactment, and scalability meaning that new resources can be easily included. On the other hand, agent [3] is a high level abstraction with intelligence to automatically execute an activity without explicitly specifying control flow. Furthermore, multiple agents can collaborate to perform a task. Therefore, using agent seems to be a good choice in enacting e-commerce process.

One of the weakness of current WFMS is its lack of interoperability due to incompatibility between underlying process modeling language (PML) and data. To solve that, we use extensible markup language (XML) [7] to model both PML and data. XML provides a universal format for data on the web. Also, XML can be extended to define a markup language for a given domain. Initially, XML was used to enhance HTML (hypertext markup language) to better describe data. Later, it is used in areas beyond that, such as e-commerce.

This agent-based process environment (APE) uses agents to exchange XML-based messages to enact e-commerce process. The process is defined by XML-based process modeling language. There are currently two kinds of agents in APE 1) client representation agent and 2) process agent. Client Representation Agent (CRA) is a stationary agent. It assumes two roles, client representative and service manager. As client representative, CRA communicates with process agent (PA) to request PA to execute XML-based process program for client, or to accept and manage the work item assigned by PA. In addition, it provides editor to edit process program, and monitoring tool to watch process enactment. It also possesses intelligence to relieve client from handling some routines. For example, CRA analyzes the content of the work assigned to locate the tools needed, and then automatically invoke the tools at the right time. As service manager, CRA executes the work assigned by PA through the external application invocation system, and reports the result of execution. The service refers to an executable file or a remote object providing CORBA services.

Process agent (PA) is a mobile agent which enacts process. Its functionality includes 1) process enactment, 2) process management and 3) process evolution support. 1) PA gets the XML-based process program from CRA, parses the process program and transforms it
to internal process model, and then interprets the model to enact activities. 2) process management provides functions to control enacting process, such as execute, pause, stop, or resume process. 3) PA supports process evolution intelligence which will be covered elsewhere.

APE communicates with PA through control message. When PA receives a control message, it will decide what to do based on the type of message:

1) Process enactment message: This message requests PA to execute process program. The message contains process program and input data.
2) Process migration message: This message instructs PA to migrate to designated place to continue enacting process. The message contains URL of the place.

Figure 1 depicts how APE enacts a process. First, the user of host A informs its CRA about which process program to enact, along with input data. Second, the CRA generates a PA to enact this process program. PA follows the instruction of the process program to migrate to host B, bringing with it the process program and the process instance state, and informs the CRA of host B to execute the first activity. After executing the activity in host B, PA migrates to host C, then to host D. Finally, PA returns to host A and reports to the CRA that the process enactment is completed.

This paper is organized as follows. Sections 2 and 3 respectively describe using extensible markup language (XML) to model process modeling language and agent communication message. Section 4 depicts the architecture of the process environment where process program processing and agent communication, among others, are discussed. Section 5 describes how the environment handles concurrent enactment. Finally, Section 6 gives the benefits of this environment.

2. XML-based Process Modeling Language

Process modeling language (PML) is used to define the work (activity) and the work flow (process). APE adopts XML-based PML. Its benefits are:

1) simple, clear, and easy to understand.
XML uses tags to describe data. By reading tags, it is very easy for others to understand the meaning of data. Also, agents can precisely enact the process by following the tags.
2) interoperability.
When WFMS enacts a process, it may need to interact with another WFMS such as requesting it to enact a sub-process. However, if the two WFMS uses incompatible PML, the two WFMS cannot communicate with each other. The nice thing about using XML is that one PML can be easily translated to another through extensible style sheet language (XSL).

The XML-based process modeling language (XPML) in APE models a process in four basic components, service, data, activity, and transition. The entire process is modeled by process template:

- **Process template** models the entire process program. It records the activities and transitions that should be executed, the used service, organization, and data type. It also uses directed acyclic graph (DAG) to represent control flow of process program.
- **Activity** models task in process program. Activity can be automatic activity or manual activity. Automatic Activity is automatically executed by computer service which can be a computer service.
application or another process template (sub-process). Manual activity is manually executed by human.

- **Transition** models control flow. It records source activity, destination activity and condition expression. Condition expression is used to evaluate whether a transition is true or false.

- **Service** models an external computer application which can be an executable file or a server that provides CORBA interface.

- **Data Type** models data type of input data of process program. It records name, type (e.g. string, integer) and source of data type. Source records how to get value of data type from input data.

Example 1 shows the structure of XPML. 

```xml
<ProcessDefinition>
  <OrganizationList>
    <Organization/>
  </OrganizationList>
  <ServiceList>
    <Service/>
  </ServiceList>
  <DataList>
    <Data/>
  </DataList>
  <ActivityList>
    <Activity/>
  </ActivityList>
  <TransitionList>
    <Transition/>
  </TransitionList>
</ProcessDefinition>
```

Example 1 : XPML Structure

3. **XML-based Agent Communication Message**

APE agents communicate with each other through agent communication messages. Currently, the messages consists of data synchronization message and evolution message. The format of the messages are represented by extensible markup language (XML).

A generic agent communication message (see

```xml
<Message Type = “Message Type” Sender = “sender id”>
  <Content>
    ...
  </Content>
</Message>
```

Example 2 : A generic agent communication message

Example 2) is represented by <Message> tag. 

```xml
<Message Type = “DS” Sender = “sender id”>
  <Content>
    <Object Name = “t1” Type = “Transition” Value = “true”/>
  </Content>
</Message>
```

Example 3 : Data synchronization message

With decentralized architecture, APE has no shared data. Each agent has a copy of data such as condition value in transition instance and executor id in activity instance. Agents have to coordinate with each other to make sure these data are consistent, so that the process can be correctly enacted. Data synchronization message serves this purpose.

The message type of data synchronization message is **DS**. Its <Content> tag contains more than one <Object> tag. <Object> tag describes object that needs to be synchronized. It has three attributes Name, Type and Value. Name describes name of object. Type describes class that the object belongs to. Value contains value of object. Example 3 shows the data synchronization message that sets condition value of transition t1 to true:

```xml
<Message Type = “DS” Sender = “sender id”>
  <Content>
    <Object Name = “t1” Type = “Transition” Value = “true”/>
  </Content>
</Message>
```
4. APE Architecture

The modules of APE are described below:

- **Process model** is a high level abstraction of process program. It models PML constructs such as activity and transition. Process management sub-system operates on the process model.

- **Communication management**. Like human, agents needs to communicate continuously with other agents to accomplish a task. Communication management subsystem provides a mechanism for agents to send and receive agent communication messages. Its implementation is based on Java Messaging System (JMS). JMS provides a uniform messaging API to drive various messaging systems by different vendors.

- **Parser** parses the process program to produce process model. Its implementation is based on the XML parser by Xerces. First, input process program is parsed to produce DOM tree. Then, by using Element Parser, ServiceList, DataList, ActivityList and TransitionList are extracted from DOM tree.

- **Process Management** is the kernel module to execute and manage process. It handles process management service and agent communication message. Details will be given shortly.

- **Agent** implements APE agent, including client representation agent (CRA) and process agent (PA) that enacts process. The implementation of CRA and PA is based on IBM Aglets Software Development Kit (ASDK) [4].

The current APE implementation consists of about 4400 lines of JAVA code in 54 JAVA classes. Among them, 15 in Process Management subsystem which is the system kernel; 6 in agent subsystem; 9 in Process Model subsystem; 11 in Parser; 6 in Communication Management subsystem; and 7 in utility and user interface.

4.1 Process Management Subsystem

Process Management subsystem is the kernel of APE. It provides 1) process management service and 2) agent communication message handling:

1. **Process Management**: Process Manager generates and manages process instance of process template. It supports services of execute, fork, migrate, suspend, and resume process. The enact Process method executes process instance. The fork Sub Process method produces another process agent (PA) to execute the process instance of parent PA. The migrate Process method migrates process instance to designated site and continue to execute process. The resume Process and suspend Process methods respectively resume and suspend process.

Process represents process instance of process template. It is represented as Directed Acyclic Graph (DAG) where vertex stores Activity Instance, and edge stores Transition Instance. Activity Instance represents instance of activity. Its private field **state** stores runtime state of activity, and **executor Id** records id of the agent that executes this activity. Transition Instance represents instance of transition. Its **condValue** records condition value of transition. Only when condition value is true can agent execute the activity pointed to by transition.

2. **Agent communication message handling**: When an agent executes a process, the agent will receive agent communication message from other agents, including data synchronization message and evolution message. Process Manager receives the messages and sends them to DataSyncMessageHandler or EvolutionMessageHandler according to message type DataSyncMessageHandler updates runtime state of process instance, such as value of transition. EvolutionMessageHandler will not be covered here.

5. Concurrent Work Flow Enactment

To enhance execution efficiency, a WFMS usually supports concurrent work flow enactment. APE adopts And-Fork to do that. And-Fork generates multiple threads from a single thread in a work flow, so that multiple activities can be concurrently enacted. For example, in Figure 2, after enacting activity A1, activities A2 and A3 should be concurrently enacted.

This mechanism is rather easy to implement in a centralized work flow engine: just forking extra threads to enact other paths in the work flow. However, it is not so in APE with decentralized architecture. The reason is that, in APE, process agent bring process program and run-time state to designated place to execute activity. When an “and-fork” occurs, the single agent has no way to migrate to multiple places at the same time. It needs multiple agents to do that. The way APE handles this is that the process agent generates multiple copies of itself. The agent enacts the first path, and the copies enact other paths.

When a “join” occurs in a work flow, that is, the flow changes from concurrent threads to single thread (e.g. activity A4 in Figure 2), only one agent is allowed to enact this join activity. Here, all the agents that enact this process instance need to coordinate to avoid the situation that the join activity is enacted by more than one agent. The way APE solves this is as follows. “Activity instance” stores the agent id that has enacted.
the activity. When an agent is going to enact an activity instance, it has to first check if another agent has already enacted the activity. If so, the agent will give up. If not, the agent will broadcast a data synchronization message telling other agents that it is going to enact the activity instance. Other agents will receive the message and remembers that some agent has already enacted the activity instance.

![Execution path of agent1](image1)
![Execution path of agent2](image2)

Figure 2: And-fork and And-join

Take Figure 2 for example, after agent 1 enacts activity A1, it forks agent 2 to enact A3, and agent 1 itself enacts A2. If agent 1 finishes A2 before agent 2 finishes A3, then agent 1 will get the right to enact A4 and it will send data synchronization message to tell agent 2 about this. When agent 2 finishes A3, it will automatically give up the right to enact A4.

6. Conclusion

This paper presents an Agent-based Process Environment (APE). APE uses agents to implement decentralized workflow management system (WFMS). Mobile agent enacts XML-based process program. Agents communicate through XML-based agent communication messages to coordinate process enactment. This APE appears to have the following benefits:

1) Reduced communication overhead.
   Since mobile agent carries process program and data to designated place to execute activity, data transfer between server and agent is no longer needed. The communication overhead is thus reduced, especially when the amount of data is large.
2) Increased Reliability.
   Process enactment is shared by multiple nodes on the net, instead of one single node as in a centralized workflow management system. Therefore, if one node is down, it will not affect enactment of other process instances. The system reliability is thus increased.
3) Increased Interoperability.
   Using XML as the formats for both process modeling language and agent communication message enhances system flexibility and interoperability. Further, by using extensible style sheet language (XSL), the process program can be easily translated to the process program of another system. The XML-based message not only eases defining new messages, but also provides a common format among agents. Thus, the system interoperability is increased.

7. Acknowledgment

This research is supported by National Science Council, Taiwan, under Grant No. NSC89-2213-E-009-022.

Reference


Accepted: Sep. 20, 2000