Cross Enterprise Business Modeling with AC Diagrams and Workflow Patterns

Shiow-yang Wu
Department of Computer Science
and Information Engineering
National Dong Hwa University
Hualien, Taiwan, R. O. C.
showyang@csie.ndhu.edu.tw

Kuo-Chang Lin
National Information Infrastructure
Enterprise Promotion Association
7F, 317, Song-Chiang Road,
Taipei, Taiwan, R. O. C.
xlen@nii.org.tw

Abstract

We proposed a framework for cross enterprise business modeling and workflow automation targeting ecommerce applications. A business process is modeled by using a new visual tool named activity-control diagram (AC diagram). Frequently occurring business procedures are captured by the adoptions of reusable workflow patterns. With formally defined semantics using distributed system theory of happens-before ordering, process behavior can be mechanically analyzed at design time. A completely specified model is automatically converted to a workflow by an iterative traversal algorithm that maps an AC diagram to an XML workflow specification. The spec can then be executed by an XML workflow engine to facilitate cross enterprise ecommerce applications.

Keywords: XML, business modeling, activity control diagram, workflow patterns, ECommerce

1. Introduction

With the explosive growth of the Internet and World Wide Web, information systems with the capabilities of handling cross enterprise business transactions over the Internet are especially in strong demands. Traditional EDI based systems can only be described as proprietary, expensive, and far from open. A flexible infrastructure and open system architecture that can fully exploit the potential of the Internet environment for business applications have been major challenges for ecommerce and information system designers. Encouragingly, the emergence of XML over the past few years opens a promising passageway out of a chaotic Internet world and into a semantic Web environment. Our goal is to employ the best of XML technologies to provide a structural framework and open architecture for the conceptual modeling and formal reasoning of business processes, the generation of workflow specifications, as well as the autoexeccution of cross enterprise ecommerce activities, all with sound theoretical basis. For business modeling, we devised a new visual tool named *activity-control diagram*(*AC diagram*) to specify business activities and the control relationships among them. A rich set of *activity notations* and *control notations* are provided for visual specification of business processes. Each notation has its corresponding XML specification. The formal semantics of the notations are precisely specified using distributed system theory of happens-before ordering to facilitate static analysis of the process behavior and to enable the automatic generation of XML workflow specification. The modeling process is further simplified by capturing frequently occurring activity and control relationships into workflow patterns. Instead of modeling everything from scratch, users can choose to synthesize a model by instantiating and connecting workflow patterns. A completely specified model is automatically converted to a workflow by an iterative traversal algorithm that maps an AC diagram to an XML workflow specification which can be executed by an XML workflow engine to facilitate cross enterprise ecommerce application integration and information exchange.

The rest of the paper is organized as follows. Section 2 provides a survey of related issues and research work. Section 3 introduces our framework and system architecture. In Section 4, we gave intuitive description and examples of the AC diagram and workflow patterns. In Section 6, we detail the formal semantics and behavior analysis of AC diagram followed by the presentation of conversion algorithm from AC diagram to XML workflow. Preliminary implementation and comparative evaluation results shown in Section 7 demonstrate the feasibility and effectiveness of our framework. Section 8 concludes the paper.

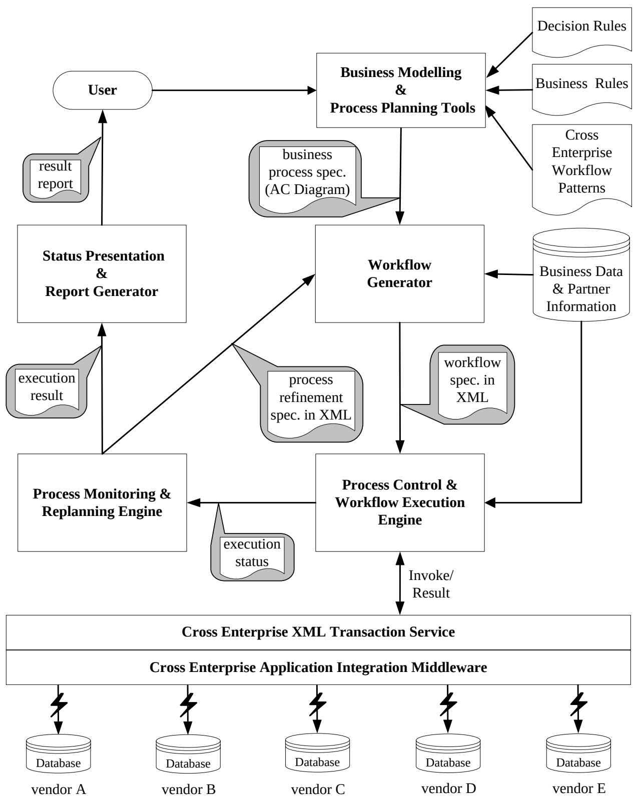
2. Related Work

The proliferation of W3C XML[14] has been phenomenal over the past few years. It has been widely used in the business and ecommerce applications. The workflow management community has adopted XML for quite some time[9, 13, 11]. In most cases, XML were used for information exchange and transformation. We took a step further to employ XML as the internal representation of our workflow specification which significantly improved the interoperability of our workflow engine and execution system. Research on inter-organizational workflow across multiple enterprise and/or heterogeneous platforms have been the focus of attention in recent years[9, 10, 11]. The concept of workflow patterns is not entirely new and has also been argued as crucial in business modeling by many researchers [1, 7]. Our work improves upon previous work by modeling generic activities in addition to common focus on control flow. Our workflow patterns are high level patterns that combine activities and controls to form process patterns while the themes of most previous work were centered around control patterns. Furthermore, we provide formal semantics of workflow control constructs based on distributed system theory of happens-before ordering. Sound theoretical basis enables us to conduct formal analysis of workflow behavior to help the users in identifying design and performance problems.

3. System Framework and Architecture

Based on the reference architecture suggested by WfMC [4], we proposed an XML-based framework for cross enterprise business modeling and ecommerce application integration. As design criteria, we want to provide a high level visual tool which is expressive enough and intuitively appealing for business modeling. Users can define their processes by drawing diagrams instead of using text-based definition language. The semantics of the visual notations must be formally specified such that the process behavior can be exactly characterized and analyzed. The cross enterprise workflow corresponding to the visual representation should be automatically generated. The system must include properly designed workflow engine as well as transaction services and middleware support for cross enterprise integration. The execution status must be monitored for its correctness and timeliness. In case of any problem, the system must allow the dynamic modification and continuing execution of the workflow. The processing status and results must be reported to the business users and/or partners.

To answer the challenges discussed above, we designed a comprehensive architecture as depicted in Figure 1. The architecture consists of seven closely interrelated modules. XML is adopted for information representation and data exchange. The *business modeling and process planing tools* are for workflow planning based on AC diagram. Users can provide business rules and decision rules for the system to follow. *Workflow patterns* can be instantiated and included as components of a larger workflow. An AC diagram is automatically converted into an XML workflow specification by the *workflow generator* which may consult a repository of business data and partner information. The XML workflow is an executable specification which can be



User
Business Modelling & Process Planning Tools
Decision Rules
Business Rules
Cross Enterprise Workflow Patterns
result report
business process spec. (AC Diagram)
Status Presentation & Report Generator
Workflow Generator
Business Data & Partner Information
execution result
process refinement spec. in XML
workflow spec. in XML
Process Monitoring & Replanning Engine
Process Control & Workflow Execution Engine
execution status
Invoke/ Result
Cross Enterprise XML Transaction Service
Cross Enterprise Application Integration Middleware
Database
vendor A
vendor B
vendor C
vendor D
vendor E

Figure 1. XML-based cross enterprise solution framework and system architecture.

processed by the *process control and workflow execution engine*. Throughout the execution, the engine may invoke arbitrary number of business operations and interact with business partners with the help of the *cross enterprise XML transaction service* which is built on top of a middle layer for cross enterprise application integration. The workflow execution status is continuously monitored by the *process monitoring and replanning engine* which may dynamically adjust and refine the workflow in case of any difficulties. The refinement specification is sent to the workflow generator to generate a refined workflow for continuous or reexecution of the workflow. All intermediate and final execution results are passed to the *status presentation and report generator* for generating summary reports. In this paper, we discuss the business modeling and process planing tools as well as the workflow generator.

4. Activity-Control Diagram

Our business modeling framework can be depicted as the layered diagram in Figure 2. At the base layer is our XML workflow specification which is directly executable by the workflow engines. The center layer is the modeling tool AC diagram consisting of activity notations and control no-

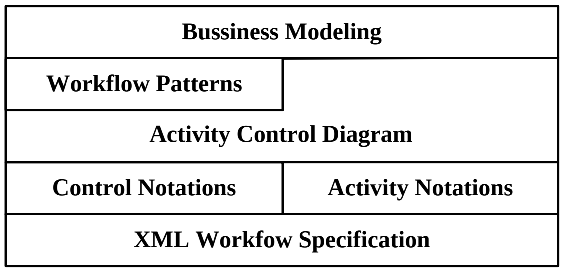


Figure 2. Business modeling framework with AC diagram and workflow patterns.

tations. Users can use them directly and/or instantiate workflow patterns. In a way similar to design patterns [6], workflow patterns are meant to be reusable structures that can be easily instantiated and interconnected. In this section, we provide intuitive description of the AC diagram with illustrating examples. Formal semantics and workflow patterns will be given in later sections.

An AC diagram consists of a set of activity notations and control notations, as well as a set of *relationships* representing the flow and dependency between them. The activity notations are used to model business activities. Instead of enumerating all kinds of activities, we propose the use of generic activities as shown in Figure 3. START and END are used for starting up and closing down a business process. DISPLAY is for reporting execution status and processing results. RETRIEVAL and INFORMATION represent one-way information flow while EXCHANGE is used to denote two-way information exchange. INTEGRATION is for multi-way information join. TRANSFORMATION is used to convert information from one form to another. DECISION is for deciding the next proper way to go among several alternatives. MODULE is a subdiagram that forms a self-contained, properly interfaced unit. CUSTOM is reserved for defining customized activities. Finally, INVOKE invokes a customized activity which is normally a partner and/or task specific application invocation.

The notations presented in Figure 4 provide a rich set of workflow control structures for defining complex business process. We provide intuitive descriptions in this section and leave the formal semantics to Section 6. Each control notation connects a set of input activities to a set of output activities and enforces a well-defined execution semantics among them. SEQUENCE is the simplest control which regulates activities one after another. OR represents a set of alternatives such that the execution can follow any one of them. A different path can be taken if the previously chosen one fails. XOR is similar to OR except for the constraint that only one of the alternatives can be chosen. AND denotes a set of activities that all must be carried out. PRIORITY is like OR with preferences explicitly

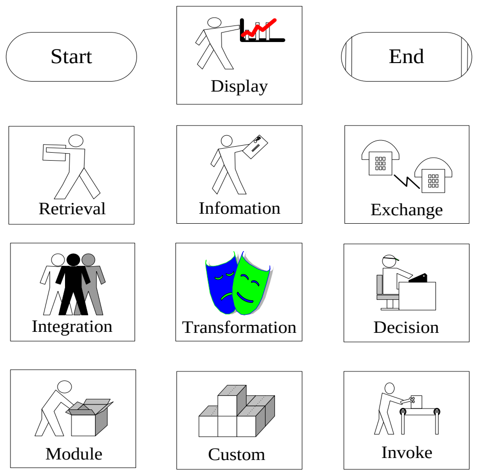


Figure 3. The activity notations.

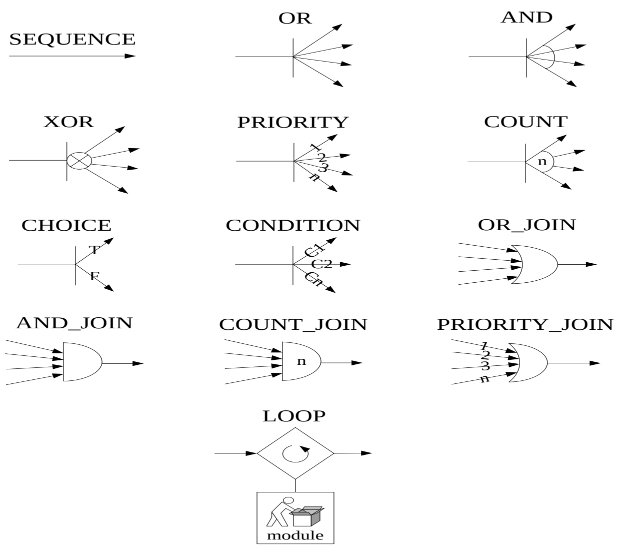


Figure 4. The control notations.

specified. Lower priority alternatives can not be started unless all higher priority alternatives have been executed and failed. COUNT is like a counting AND which denotes the requirement to execute at least n activities out of all alternatives. The number n must be smaller or equal to the number of alternatives. CHOICE denotes a conditional branching where the choice of an alternative is determined by a boolean condition. CONDITION is a generalized CHOICE where the selection of alternatives is determined by generalized conditions rather than a boolean condition. It also has the property of AND since all branches with successful conditions must be executed. Except for the SEQUENCE, all seven notations above are collectively referred to as *split structures* that connect one input activity to multiple output activities. The next set of four notations are known as

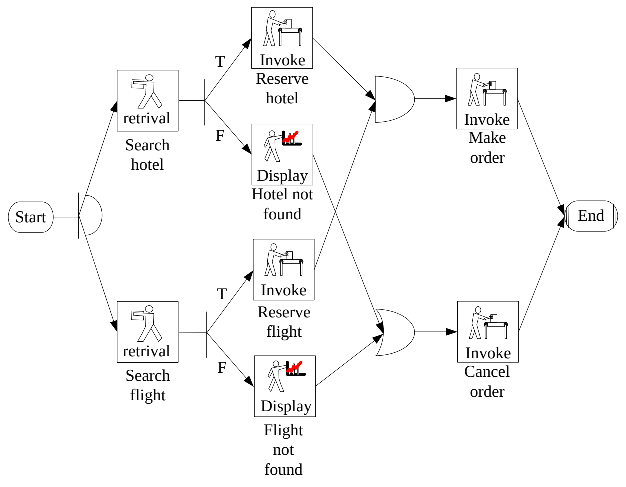


Figure 5. A travel planning AC diagram.

join structures which connect multiple input activities to a single output activity. OR_JOIN denotes that the success of any input activity is good enough to trigger the output activity. AND_JOIN means that all input activities must success before starting the output activity. COUNT_JOIN is like a counting AND_JOIN. PRIORITY_JOIN denotes that the output activity can only be triggered by an input activity of priority i if all higher priority input activities have been tried and failed. Finally, LOOP is used to model the repeated execution of a sub diagram represented as a module. The loop continues as long as the loop condition is satisfied.

The notations are designed to be intuitive enough for non-IT users to understand and use properly. Figure 5 is an example of a travel planning AC diagram. For a business trip to be arranged successfully, we must do both a flight search and a hotel search. If proper flight and hotel can be found, we can proceed with flight and hotel reservation. If both activities end without any problem, we can go ahead to make the order and finish the job. Otherwise, we may need to cancel existing order and end the flow.

5. Workflow Patterns

Even though the activity and control notations are simple enough to be used directly, some structures do show up repeatedly in the business modeling process. We capture these frequently occurring structures into workflow patterns as the building blocks for complex workflow design. A workflow pattern is a generic set of activity and control notations structured according to a frequently occurring business subprocess. Patterns are designed to be easily conceivable and reusable such that a user can quickly identify their structural meaning and recognize their usage. We have designed a rich set of workflow patterns for various situations. Because of the space limit, we present two representative patterns in this section.

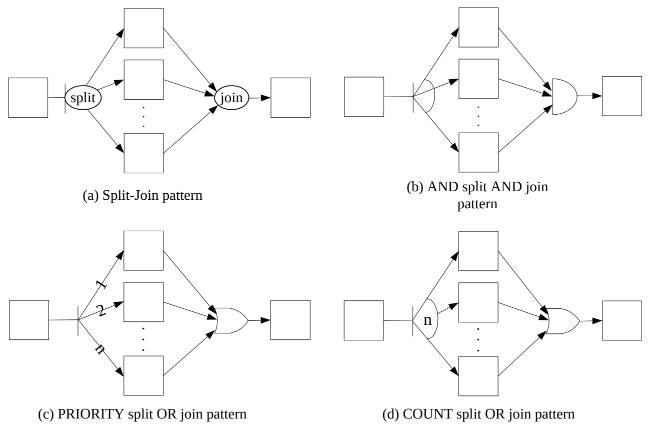

split
join
(a) Split-Join pattern
(b) AND split AND join pattern
1
2
n
(c) PRIORITY split OR join pattern
n
(d) COUNT split OR join pattern

Figure 6. The Split-Join patterns.

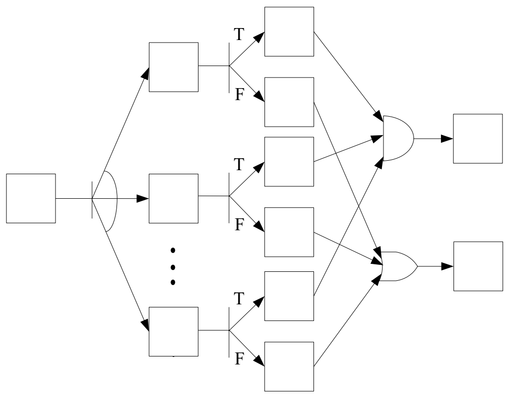


Figure 7. The And-Or pattern.

The **Split-Join Patterns** denote a family of patterns that start with a split type control and end with a join type control as depicted in Figure 6(a). Several split and join control notation pairs can be combined to form meaningful split-join patterns. For example, Figure 6(b), (c), and (d) denote an AND split AND join, PRIORITY split OR join, and COUNT split OR join pattern, respectively. An AND split AND join pattern denotes the situation where a number of activities that must all be carried out successfully. A PRIORITY split OR join pattern corresponds to the situation where a number of alternative activities can be taken to satisfy the job requirement. But the user prefers some activities over the others if possible.

Another example is the **And-Or Pattern** as depicted in Figure 7. In here, we capture the situation that a set of subtasks must all be done successfully to complete a task where the successfulness of each subtask is determined based on the execution result of each subtask. Any failure of the subtasks will result in the failure of the original task. As a matter of fact, the travel planning example presented in Figure 5 is designed based on the and-or pattern.

6. Semantics, Analysis and Conversion

In this section, we formally define the semantics of our notations based on distributed system theory of happens-before ordering[8]. In the following definitions, we use the notation $a \rightarrow b$ to denote the ordering that a happens-before b. We also use $\mathcal{A}$ and $\mathcal{C}$ to denote the set of activity notations and control notations, respectively.

Definition 1 (Preliminaries)
Several notations are needed throughout the definitions.

$start(a)$	is the start time of the activity a.
$finish(a)$	is the end time of the activity a.
$exec(a)$	is to execute the activity a.
$repeat(m, c)$	is to repetitively execute module m whenever the condition c is true.
$success(a)$	to denote that activity a has been successfully completed.
$fail(a)$	to denote that the execution of activity a has failed.

Definition 2 (AC Diagram)
An AC Diagram is a triple (A, C, R) where $A = \{a_1, a_2, \ldots, a_n | \forall i, a_i \in \mathcal{A}\}$ is the *activity set*, $C = \{c_1, c_2, \ldots, c_m | \forall i, c_i \in \mathcal{C}\}$ is the *control set*, and $R = \{(I, c, O) | I \subseteq A, O \subseteq A, c \in C\}$ is the *relationship set*.

Definition 3 (Activity)
An *activity* is a tuple $(a_{id}, atype, P, c, t)$ where

a_{id}	is the unique identifier of the activity.
$atype$	is the type of the activity as depicted in Figure 3.
P	is the set of parameters.
c	is a constrain expression.
t	is an optional deadline.

Definition 4 (Control)
A control is a tuple $(c_{id}, ctype, I, O, d, t)$ where

c_{id}	is the unique identifier of the control.
$ctype$	is the type of the control as depicted in Figure 4.
I	is the input activity set.
O	is the output activity set.
d	is an ordered set of condition(s) (for CHOICE, CONDITION, and LOOP) or a count (for COUNT and COUNT_JOIN).
t	is an optional deadline.

Definition 5 (Ordering)
To adopt the happens-before ordering, we define a notation to denote the successive execution of two activities. More specifically, the fact that the execution of activity o should follow the completion of activity i is defined and denoted as

$$Order(i, o) = finish(i) \leq start(o) \wedge \\ \forall x \in A, o \rightarrow x \Rightarrow start(o) < start(x)$$

Definition 6 (SEQUENCE)
A SEQUENCE is a control of the form $(c_{id}, \text{SEQUENCE}, \{i\}, \{o\}, null, t)$ with the following semantics.

$$success(i) \Rightarrow exec(o) \wedge Order(i, o)$$

Definition 7 (OR)
An OR is a control of the form $(c_{id}, \text{OR}, \{i\}, \{o_1, \ldots, o_n\}, null, t)$ with the following semantics.

$$success(i) \Rightarrow \exists k, 1 \leq k \leq n, exec(o_k) \wedge Order(i, o_k)$$

Definition 8 (AND)
An AND is a control of the form $(c_{id}, \text{AND}, \{i\}, \{o_1, \ldots, o_n\}, null, t)$ with the following semantics.

$$success(i) \Rightarrow \forall k, 1 \leq k \leq n, exec(o_k) \wedge Order(i, o_k)$$

Definition 9 (XOR)
An XOR is a control of the form $(c_{id}, \text{XOR}, \{i\}, \{o_1, \ldots, o_n\}, null, t)$ with the following semantics.

$$success(i) \Rightarrow \\ \exists k, 1 \leq k \leq n, exec(o_k) \wedge Order(i, o_k) \\ \wedge \forall j \neq k, 1 \leq j \leq n, \neg exec(o_j)$$

Definition 10 (PRIORITY)
A PRIORITY is a control of the form $(c_{id}, \text{PRIORITY}, \{i\}, \{o_1, \ldots, o_n\}, null, t)$. Let o_{n+1} be a dummy action such that $success(o_{n+1}) = true$. Then the semantic can be described as follows.

$$success(i) \Rightarrow \\ \exists k, 1 \leq k \leq n+1, exec(o_k) \wedge Order(i, o_k) \\ \wedge success(o_k) \wedge (\forall j, 1 \leq j < k, exec(o_j) \wedge fail(o_j))$$

Definition 11 (COUNT)
A COUNT is a control of the form $(c_{id}, \text{COUNT}, \{i\}, O = \{o_1, \ldots, o_n\}, count, t)$ with the following semantics.

$$success(i) \Rightarrow \exists S \subseteq O, |S| = count \wedge \\ (\forall j \in S, exec(j) \wedge Order(i, o_j))$$

Definition 12 (CHOICE)
A CHOICE is a control of the form $(c_{id}, \text{CHOICE}, \{i\}, \{o_T, o_F\}, d, t)$ with the following semantics.

$$success(i) \Rightarrow \\ ((d = true) \Rightarrow exec(o_T) \wedge Order(i, o_T)) \\ \vee ((d = false) \Rightarrow exec(o_F) \wedge Order(i, o_F))$$

Definition 13 (CONDITION)
A CONDITION is a control of the form $(c_{id}, \text{CONDITION}, \{i\}, \{o_1, \ldots, o_n\}, \{d_1, \ldots, d_n\}, t)$ with the following semantics.

$$success(i) \Rightarrow \forall k, 1 \leq k \leq n, \quad (d_k = true) \Rightarrow exec(o_k) \wedge Order(i, o_k)$$

Definition 14 (OR_JOIN)
An OR_JOIN is a control of the form $(c_{id}, \text{OR\_JOIN}, \{i_1, \ldots, i_n\}, \{o\}, null, t)$ with the following semantics.

$$(\exists k, 1 \leq k \leq n, success(i_k) \wedge k = a) \Rightarrow exec(o) \wedge Order(i_a, o)$$

Definition 15 (AND_JOIN)
An AND_JOIN is a control of the form $(c_{id}, \text{AND\_JOIN}, \{i_1, \ldots, i_n\}, \{o\}, null, t)$ with the following semantics.

$$(\forall k, 1 \leq k \leq n, success(i_k)) \Rightarrow exec(o) \wedge \forall j, 1 \leq j \leq n, Order(i_j, o)$$

Definition 16 (COUNT_JOIN)
A COUNT_JOIN is a control of the form $(c_{id}, \text{COUNT\_JOIN}, I = \{i_1, \ldots, i_n\}, \{o\}, count, t)$ with the following semantics.

$$(\exists S \subseteq I, |S| = count \wedge \forall j \in S, success(i_j)) \Rightarrow exec(o) \wedge \forall j \in S, Order(i_j, o)$$

Definition 17 (PRIORITY_JOIN)
A PRIORITY_JOIN is a control of the form $(c_{id}, \text{PRIORITY\_JOIN}, \{i_1, \ldots, i_n\}, \{o\}, null, t)$. Let i_{n+1} be a dummy action such that $success(i_{n+1}) = true$. Then the semantic can be described as follows.

$$(\exists k, 1 \leq k \leq n+1, success(i_k) \wedge k = a \wedge (\forall j, 1 \leq j < k, fail(i_j))) \Rightarrow exec(o) \wedge Order(i_a, o)$$

Definition 18 (LOOP)
A LOOP is a control of the form $(c_{id}, \text{LOOP}, \{i\}, \{o, m\}, d, t)$ where m is a module. The semantics can be described as follows.

$$success(i) \Rightarrow ((d = true) \Rightarrow repeat(m, d) \wedge Order(i, m) \wedge exec(o) \wedge Order(m, o)) \vee ((d = false) \Rightarrow exec(o) \wedge Order(i, o))$$

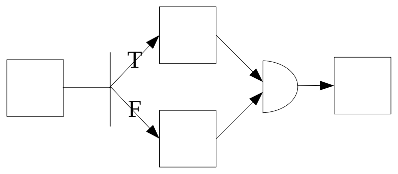
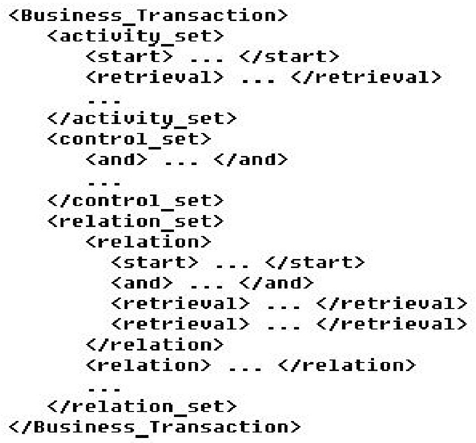


Figure 8. An example of the ExSplit-ConJoin Inconsistencies.



```
<Business_Transaction>
   <activity_set>
      <start> ... </start>
      <retrieval> ... </retrieval>
      ...
   </activity_set>
   <control_set>
      <and> ... </and>
      ...
   </control_set>
   <relation_set>
      <relation>
        <start> ... </start>
        <and> ... </and>
        <retrieval> ... </retrieval>
        <retrieval> ... </retrieval>
      </relation>
      <relation> ... </relation>
      ...
   </relation_set>
</Business_Transaction>
```

Figure 9. The XML skeleton of an AC diagram.

With clearly defined semantics, formal behavior analysis can be naturally conducted. For example, by following the happens-before ordering, we can identify the longest or critical path in the diagram to estimate the time required for executing the workflow. Another important type of analysis is to discover semantic inconsistencies in an AC diagram. We have identified several classes of inconsistencies. For example, the **ExSplit-ConJoin Inconsistencies** are the patterns that connect an exclusive split (CHOICE, XOR or PRIORITY) with a concurrent join (AND_JOIN or COUNT_JOIN with count > 1) since an exclusive split only allows one alternative to be executed while a concurrent join needs more than one inputs to be successfully completed. Figure 8 is an example that connects a CHOICE with an AND_JOIN which is clearly inconsistent since it is impossible for both inputs of the AND_JOIN to succeed. The inconsistency analysis can help the users in identifying design problems.

After a business process has been modeled as an AC diagram, our next goal is to automatically convert the diagram into an XML workflow specification. This is done by first giving each notation (activity and control) its corresponding XML specification, and then design an algorithm to perform the conversion. Because of the space limit, we only present the XML skeleton of the AC diagram (denoted as a Business_Transaction) in Figure 9.

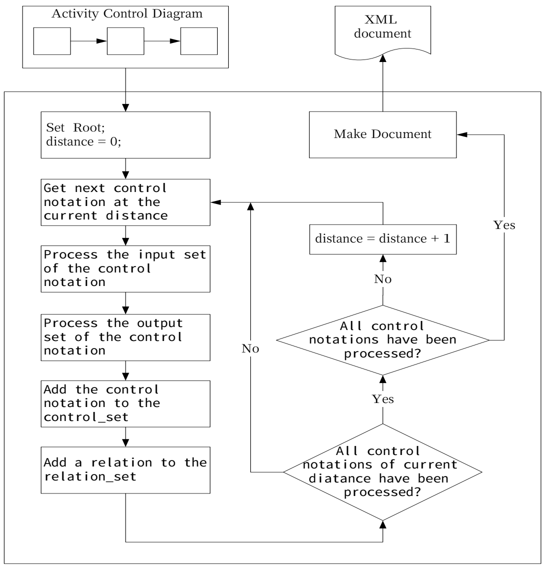

Activity Control Diagram
XML document
Set Root; distance = 0;
Make Document
Get next control notation at the current distance
distance = distance + 1
Yes
No
Process the input set of the control notation
Process the output set of the control notation
All control notations have been processed?
No
Yes
Add the control notation to the control_set
Add a relation to the relation_set
All control notations of current diatance have been processed?

Figure 10. AC diagram to XML workflow specification conversion algorithm.

The algorithm for converting an AC diagram to its corresponding XML workflow specification is presented in Figure 10. The algorithm takes an essentially breadth-first approach to process the activity and control notations level by level. For each control notation, the input set is processed first, followed by the output set, then finally the control itself. After the processing of each control notation, a relation is added to the relationship set. The algorithm then proceeds to the next control at the same level or advances to the next level if all controls at the current level have been processed.

7. Prototype and Evaluation

To evaluate the usefulness and performance of our approach, we have implemented a prototype workflow engine using Java, DOM, XQuery, and XML related APIs. The organization of the engine is depicted in Figure 11. The *Engine Control Center* is responsible for taking the XML workflow specification as input, control the execution of the workflow, and generate the final execution report. The XML specification is validated by the *DOM Parser* to generate the corresponding DOM tree for further processing. The *Control Processor* is the key component for maintaining the proper semantics of the control notations. The *Activity Processor* takes an activity node with associated parameters and invokes proper procedures for executing the activity. Finally, the *Status Monitoring & Recording* component is to monitor the workflow execution status and record intermediate results for other components to employ.

With the prototype implementation, we compare the AC diagram with two commercially available workflow management tools, the Agentflow from FlowRing Tech Corp[5] and the EZ-Modeler from DynaFlow Inc.[3]. Agentflow employs a proprietary process definition tool named *Process Designer Engineer*(PDE) while EX-Modeler uses the popular business modeling tool *Baan IV DEM*[12]. We therefore conduct a preliminary comparison of AC diagram with PDE and DEM by using all three of them in modeling the travel planning example in Figure 5. The result is shown in Table 1. In general, the modeling primitives of DEM and PDE can all be covered by the activity and control notations of AC Diagram but not vise versa. For example, the PRIORITY, COUNT_JOIN, and PRIORITY_JOIN control notations do not have counterparts in either DEM or PDE. Another advantage of the AC Diagram is the use of XML as workflow and data representation formats. The adoption of XML helps us significantly in the design of workflow engine for cross-enterprise application integration.

For performance evaluation, we designed a scalable AC

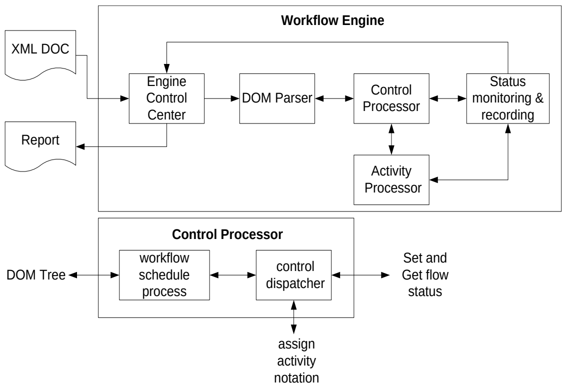


Figure 11. The Workflow Engine.

	AC Diagram	DEM	PDE
Theoretical basis	happen-before	high level Petri-net	OOAG
No. links	12	22	14
No. activities	10	14	10
Activity and control	yes	yes	no
Types of controls	plenty	few	few
Types of activities	plenty	few	N/A
Representation	XML	proprietary	proprietary
Behavior analysis	happens-before	Petri-net	proprietary
Model complexity	simple	complex	simple

Table 1. Comparison of AC Diagram, DEM, and PDE.

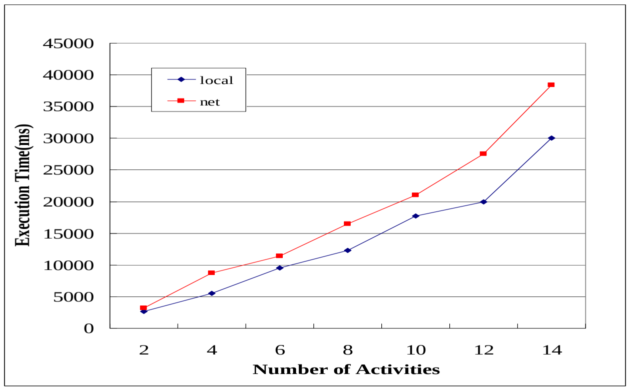
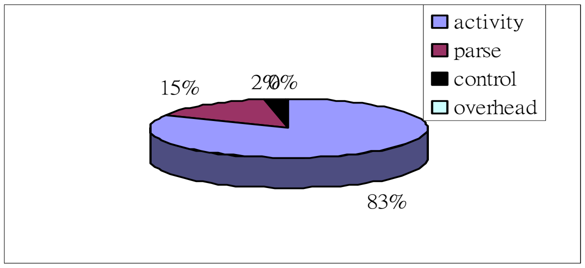


Figure 12. The performance of the workflow engine on local machine vs over network.



activity	parse	control	overhead
82.83%	14.52%	2.37%	0.28%

Figure 13. The execution time percentage on different parts of the workflow engine.

diagram and measured the execution time on a local vs network environment. For local execution, all activity and control invocations were done on the same machine. For network execution, all activities were carried out on remote machines. The purpose was to simulate cross-enterprise execution. Figure 12 shows that the network execution time was close to the local execution time which demonstrates the efficiency of the workflow engine. We also measured the percentage of time spent on different parts of the workflow and the overhead. We can observe in Figure 13 that over 85% of time were spent on real work (i.e. activities and controls) and less than 15% of time was spent on parsing XML. The remaining overhead is negligible.

8. Conclusions and Future Work

We have proposed an XML-based framework for cross enterprise electronic commerce applications. AC diagram and workflow patterns are used for business modeling and workflow automation. Happens-before ordering is employed for specifying the formal semantics of the notations and for behavior analysis. A prototype workflow engine is designed and implemented to evaluate the feasibility and performance of our approach. Comparative evaluation and experimental results show that AC diagram and workflow patterns are expressive, easy to use, and efficiently implementable with very low execution overhead.

One of the problem we encountered is that our notations are generic and therefore need to be instantiated according to different business semantics. For a fixed implementation of the workflow engine, we need to pass enough parameters to properly instantiate a generic notation to the right business semantics. To this end, we plan to design a new approach of executing the workflow by using dynamic agents[2] that can be dynamically associated with different knowledge bases for changing business environments. We also plan to develop transaction management mechanism such that a business process can be designated to execute in transaction mode with ACID guarantee.

References

[1] W. Aalst, A. van der, B. Hofstede, and A. Kiepuszewski. Advanced workflow patterns. In *7th Intl. Conf. on Cooperative Information Systems (CoopIS 2000)*, pages 18–29, 2000.

[2] Q. Chen, P. Chundi, U. Dayal, and M. Hsu. Dynamic-agents. *Intl. J. on Cooperative Information Systems*, 1999.

[3] DynaFlow Inc. EZ-Process Suite, 2004.

[4] L. Fischer, editor. *The Workflow Handbook 2004*. The Workflow Management Coalition, 2004.

[5] Flowring Technology Corp. Agentflow- Product White Paper, 2004.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[7] B. Kiepuszewski, A. Hofstede, and W. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, 2003.

[8] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, Jul 1978.

[9] K. Lenz and A. Oberweis. Modelling interorganizational workflows with XML nets. In *Proc. 34th Hawaii Intl. Conf. on System Sciences*, Jan. 2001.

[10] J. Meng, S. Su, H. Lam, and A. Helal. Achieving dynamic inter-organizational workflow management by integrating business processes. In *Annual Hawaii Intl. Conf. on System Sciences (HICSS'02)*, 2002.

[11] G. Shegalov, M. Gillmann, and G. Weikum. XML-enabled workflow management for e-services across heterogeneous platforms. *The VLDB Journal*, 10(1):91–103, 2001.

[12] SSA Global. Baan IV DEM, 2004.

[13] W. van der Aalst and A. Kumar. XML based schema definition for support of inter-organizational workflow. *Information Systems Research*, 14(1):23–47, 2003.

[14] W3C. The World Wide Web Consortium - Extensible Markup Language (XML), 2004.