Semantic Relation Based Exception Detection in Workflow Systems

LI Hai-bo[†,††] *, ZHAN De-chen*[†] *,XU Xiao-fei*[†]

[†]*Centre of Intelligent Computing of Enterprises, School of Computer Science&Technology, Harbin Institute of Technology, Harbin 150001, China*
[††]*Engineering College, Northeast Agricultural University, Harbin 150030, China*

Summary
Due to foreseen or unforeseen situations, deviations of workflow processes from their specifications are unavoidable. To reduce exceptions at run-time, exceptions should be found out first. An approach to detect exceptional path of workflow is presented. The proposed method analyzes the semantic relations between business activities first, and then describes them as data dependency rules. Using these rules, as a kind of semantic supplement to workflow control rules, the given exception detection algorithm can search all anticipative paths of workflow; hence identify all exceptional paths. Without changing workflow schema, the semantic rules can help to avoid exceptional paths, so as to reduce the possibility of executing ineffectual business processes. In addition, the method keeps the rationality of business process logically. Finally, a practical example is given to explain the exception detection algorithm.

Key words:
workflow; exception detection; semantic relation; data dependency rule; control rule

1. Introduction

A workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [1]. At build time, business processes are defined in workflow schema. Execution order of activities forms workflow path at run time. However, due to foreseen or unforeseen situations, such as system malfunctions due to failure of physical components, database broken down, data error or changes in business environment, deviations of those workflow processes from their specifications are unavoidable [2-6]. The deviations are often called exception. What is exception depends on what we plan and what we can achieve, between which there are always conflicts [7]. Since it is impossible or time-consumed for developers to predefine all potential exceptions in a workflow, especially those special cases which seldom happen, exceptions occur frequently during the execution of a business process. In addition, if similar exceptions frequently arise, they are always added into workflows schema as foreseen exceptions. This results in a more and more complex schema. Therefore, a WFMS (Workflow Management System) has to provide exception mechanisms to deal with constant amendment and other models related with it.

Among many causes of exception, one arises from semantic relation between activities in workflow. For example, in an equipment maintenance process, some broken-down spare parts must be relegated to other companies, and therefore if workflow system knows this knowledge as early as possible, the process involving dispatching, picking and repairing, which is doomed to failure in future can be avoided, and can be turned to the activity 'repair on commission' directly. This type of exception occurs all the same although workflow schema is correct. So semantic exception in workflow systems in our view refers to unsuccessful situations caused by lacking, malapropos or illogic semantic relation.

An information system should be able to fulfill three functions about exception: to know, detect and resolve, among which to detect semantic exception is one of the important tasks in exception handling. The objective of semantic exception detection is to capture failure in workflow systems. We believe that capturing semantic exception should depend on semantic characteristic. Many researchers have discussed related methods. In Ref. [7], exception detection can be achieved by supervising the external inputs and outputs of workflow system components, and comparing their behavior with the specified behavior of the system. In Ref. [8], exception detection can be treated as programming languages. All these approaches have less operability or more complexity. Exception handling can be described in workflow model [9], but these can lead to more complex schema that is difficult to understand. So it is significant to detect exception in future workflow path without modifying workflow schema before successor activities are executed, especially in those processes which consume much more manpower and material resources.

This thesis analyzes the basic feature of workflow first, and mines data dependency relationship between business activities. The data dependencies, as a kind of supplement to workflow control rule, are employed by the given exception detection algorithm to detect unsuccessful paths beforehand in workflow schema. Because the proposed exception detection approach is based on the basic feature of workflow, it can be applied in various

Manuscript received February 25, 2006.
Manuscript revised February 28, 2006.

environments. Employing the approach, not only complexity and increasing maintenance cost of workflow model caused by modification can be avoided, but also rationality of business process logically can be kept.

The rest of the paper is organized as follows. Section 2 analyzes foundational features of workflow. In section 3 an intelligent research algorithm is explained and in section 4, a practical example is given to verify the effectiveness of this approach. Finally, section 5 shows the conclusion.

2. Foundational Features of Workflow

A workflow schema is the formal description of a business process, which is operated and executed automatically by WFMS. A WFMS consists of two parts: modeling and executing. Workflow specification given in the modeling phase defines attributions related with run-time data, such as input and output data. At run time, execution of activities forms paths, i.e. workflow, following logical relation and dependency rules defined in workflow specification. The WFMS coordinates business activities in a business process described in workflow specification. Logical control relation between activities depends on four types of control structures, namely sequence structure, parallel structure, selective structure and iterative structure, which are defined by Workflow Management Coalition (WfMC) [1]. Besides control dependency between activities, there exists data dependency, for example, data input of one activity depends on data output of other activities. These basic features, as foundation of workflow, are defined by workflow specification. We give some definitions about workflow specification and these features first.

Definition 1(Workflow Specification) Workflow specification *ws* is a triple *ws*=(*N*, *F*, *R*), where N $=\{n_1,n_2,\ldots,n_n\}$ is a set of activity nodes, F=$\{ f_i \mid f_i=<n_s,n_t>, n_s,n_t \in \mathrm{N} \}$is a set of path between nodes. R=$\{DR,CR\}$ is a set of dependency rules, where *DR* represents a set of data dependency rules and *CR* is a set of control rules. There exist only one initial activity n_s and one final activity n_e.

Path $<n_s,n_t>$ represents a partial order between n_s and n_t. This partial order can be denoted as $n_s \prec n_t$. In fact, control nodes should be involved in definition 1. Here we ignore control nodes due to its unimportant effect.

Partial order between activities depends on control rule. In information system, dependency rules act as function or predication, including data dependency rule and control rule. The former refers to dependency based on data at run time, while the later refers to dependency based on logical control relation.

Definition 2(data dependency) Let $a_i.D$ and $a_j.D$ be data operated by activity a_i and a_j respectively. If producing $a_j.D$ needs to read $a_i.D$, say that there exists data dependency between activity $a_i.D$ and $a_j.D$. If producing $a_j.D$ must satisfy $P_i(a_i.D)$=TRUE, then P_i is data dependency rule between $a_i.D$ and $a_j.D$, denoted as $P_i(a_i.D)\rightarrow a_j.D$ where $P_i \in$DR is predication of dependency rule, while if there does not exist data dependency relation, denoted as $P_i(a_i.D) \nrightarrow a_j.D$.

Exceptions caused by semantic deviations need to be caught by analyzing semantic relation between activities, such as data dependency. The data dependency between activities is composed of two categories: explicit dependency and implicit dependency. The former refers to the relation of data input and output between different activities, while the later characterizes that business activities are triggered by others depending on some global data or changing states. In the case of implicit dependency, data is not transferred explicitly, for example, the state of an order list is changed after being authorized, and the activity *purchase* is triggered. Timing triggering in workflow system is another example of implicit dependency.

The given data dependency is concerned with relationship between different activities. For data inside the same activity, the dependency rule is guaranteed by itself. So we say that $P_i(a_i.D)$ is precondition of executing a_j if $P_i(a_i.D)$=TRUE is met when executing a_j. In order to discuss expediently, we suppose that data set used to write does not intersect with each other for avoiding data writing conflicts.

Definition 3(control dependency) For $\forall a_i,a_j$, if a partial order can be determined between a_i and a_j , according to $P \in CR$, controlled by four basic control structures, we say that there exists control dependency between a_i and a_j , denoted as $a_i \xrightarrow{P} a_j$. *P* is called control dependency rule, for short, control rule.

3. Exception Detection of Workflow

3.1 Some Properties of Workflow Schema

Execution of activities in logistic order forms a flow at run time, which can be described by directed graph. In the graph, nodes represent activities, and directed arcs denote logistic order of nodes, i.e. control dependency between activities. The formal model is transformed to a middle language which can be recognized by computer, and is called workflow specification generally. Directed arc is also used to describe data flow, i.e. data dependency between activities. Depending on different significations of arc, we define two types of directed graphs. If arc represents control flow, the graph is called control dependency graph, denoted as G_{Ctrl}, while if data flow, the graph is called data dependency graph, denoted as G_{Data}.

The two ‘flow’s formed by data dependency and control dependency between activities are not always consistent. In some situations, after an activity node completes, it offers data not only to next node which has control dependency with it, but also to other nodes without direct control dependency with it, while the ‘other nodes’ have not reasonable control dependency relation with the node which has been completed, so that this situation can not be described in G_{Ctrl} [10]. In Fig.1, activity *c* is not directly connected with node *a*, but there exists data dependency between them, as dotted line. Though G_{Ctrl} and G_{Data} are not always consistent, the following properties are useful.

Property 1 If control dependency exists between nodes in G_{Ctrl}, data dependency exists between them also.

Proof. Give two nodes *a* and *b*, and control dependency between them, having order $a \prec b$ or $b \prec a$. Assume $P_i(a.D) \not\rightarrow b.D$, i.e. input data of *b* is not read from output data of *a*, then there exists uncertain logistic order between them, i.e. none control dependency between them is available. Therefore, the reasoning result conflicts with the assumption $a \prec b$ or $b \prec a$.

If node *c* is a successor of node *b*, showed in Fig.2(a), node *b* must provide some data to node *c*, i.e. there must be data dependency between node *b* and *c*, or node *c* need not be successor of *b*. For example, activity b and c can execute in parallel theoretically, see Fig.2(b).

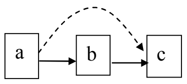

Fig.1 data dependency relation between activities

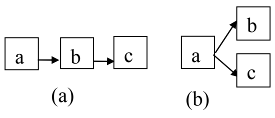

Fig.2 control dependency relation between activities

Property 2 G_{Ctrl} of workflow is a subgraph of its G_{Data}, and they have equal amount of vertexes.

Proof. Let amount of vertexes in G_{Ctrl} and G_{Data} be *m* and *n* respectively. If $m<n$, there exists at least an activity *a*, and its position in G_{Ctrl} is uncertain, that is, having no certain partial order with other nodes. This conflicts with definition 3. If m>n, it does not satisfy property 1. So they must have equal amount of vertexes. Let amount of arcs of G_{Ctrl} and G_{Data} be *s* and *t* respectively. Assume s>t, this does not satisfy property 1, either. So we get $s \le t$, i.e. G_{Ctrl} of workflow is a subgraph of its G_{Data}.

3.2 Exception Detection Algorithm of Workflow

Constrained by a set of control dependency rules *CR*, execution of activities forms a path. Property 2 concludes that data dependency rules, as a kind of semantic supplement, enrich control rules of routing if they are considered before routing. Control rules take effect only when the last activity completes, and its next activity will be started. Whereas, data dependency rules can be applied at any run time as long as activities that produce output data complete. By using exception detection algorithm to search all possible paths before starting an activity, a WfMS can detect those consequentially unsuccessful paths in future. These paths are exceptional paths. This is the core of ideology for exception detection in workflow. In addition, some data related with data dependency rule take effect only when its activity completes. The following are some definitions.

Definition 4 (**path**) In sequence *p*, where $p=<n_1,\ldots,n_t>, n_j \in N, j=1,\ldots,t$, if $<n_i,n_{i+1}> \in F$, $i=1,\ldots,t-1$, say *p* is a path in workflow specification *ws*.

Definition 5 (**Available Data**) Activity node *n* completes, say *n.D* is available data, denoted as *Available* (*n.D*)=TRUE.

Definition 6 (**Exceptional path**) On path $p=<n_1,\ldots,n_t>$, for $\forall n_j \in N, j=1,\ldots,t$, and all data dependency relation $P_i(n_i.D) \rightarrow n_j.D$, $P_i \in$ DR, if $\exists P_i(n_i.D)$ = FALSE and *Available* ($n_i.D$) = TRUE, say that *p* is exceptional path. Exceptional path is always denoted as *Valid*(*p*), which returns a boolean variable. If *Valid* (*p*) = FALSE, *p* is exceptional path.

The definition of exceptional path shows that data operated in an activity will have impact on its next activity in the same path when the activity has completed. This impact is characterized by data dependency rules, which provide more semantic rules to identify exceptional paths in workflow. In order to detect exceptional paths, exception detection algorithm search G_{Ctrl} first. We consider G_{Ctrl} as DAG (Directed Acyclic Graph) [11]. Soundness of workflow schema requires that it be reachable from the start node to any other nodes and further more, that it be also reachable from any reachable nodes to the end node, and that there be only one start node and one end node in a workflow schema [11]. In G_{Ctrl}, all workflow execution sequences have only one entrance because in workflow graph it is the only node which has not any incoming arcs, but has outgoing arcs. Similarly, in workflow schema there is only one end node which has not outgoing arcs, but has incoming arcs. Starting from any node, the execution path must be able to end at node n_e [12].

We adopt traditional data structure - adjacent table to store G_{Ctrl}. Let *adjlist* be adjacent table of GCtrl, *adjlist[i]* be a list headed by node n_i, node $n_{i1},n_{i2},\ldots$, and n_{im} be *m* activities adjacent to n_i. Let *P* and *temp* be array list used to store path, *Q* be node list. Define the following basic functions: *FIRSTADJ*(G_{Ctrl} ,n_i) represents the first adjacent of node n_i, *NEXTADJ*(G_{Ctrl}, n_i ,n_j) represents the next adjacent node of adjacent node n_j of node n_i, *EnQueue*(*Q*,*v*) and *v*=*DeQueue*(*Q*) represent the function of inputting node *v* to list and outputting node *v* respectively, GetPath(*P*,*v*) represents the function of getting all paths whose last node is *v*, *PutPath*(*p*,*v*) represents putting node *v* to the end of each path *p*, and *COPYLIST*(*P1*,*P2*) represents copying all paths from *P2* to *P1*.

Algorithm 1 *AllPath*(G_{Ctrl}, v_0) - seek all paths from any node v_0 to end node n_e.

```
Input: G_Ctrl and any node $v_0$ in G_Ctrl, $v_0 \neq n_e$
Output: set of all paths $Path=\{p_1,...,p_k\}$, between node $v_0$ and $n_e$
Q=$\emptyset$; EnQueue(Q, $v_0$); Path [1]=< $v_0$>;
While Q$\neq\emptyset$ do
    v = DeQueue(Q);
    w = FIRSTADJ (G_Ctrl , v);
    temp = $\emptyset$;
    temp = GetPath(Path , v);
    While w$\neq\emptyset$ do
        PutPath (temp , w);
        COPYLIST(Path , temp);
        EnQueue(Q , w) ;
        NEXTADJ(G_Ctrl , v , w);
    Return Path;
```

In fact, algorithm 1 is a Breadth First Search(BFS) in G_{Ctrl}, and gets out the set *P* of paths, and $P\neq\emptyset$ as long as $v_0\neq n_e$, whose integrality is guaranteed by Ref.[12]. G_{Data} of workflow is a semantic supplement to G_{Ctrl}. By property 2, given a path, algorithm 2 is employed to detect its validity. Define a basic function first: *P*=*GetRule*($n_j.D$) represents getting the set of data dependency rules which are depended by $n_j.D$.

Algorithm 2 *ExceptionChecking*(*p*) – detecting exceptional path

```
Input: $p=<n_1,...,n_t>$
Output: boolean value
j=1;
while j<t do
  P = GetRule($n_j.D$);
    While P$\neq\emptyset$ do
      P $\leftarrow$P – {$\forall P_i | P_i \in P$ };
      If $P_i(n_i.D)$ =FALSE and Available ($n_i.D$)=TRUE then
      return FALSE;
      j=j+1;
return TRUE;
```

At any time when an activity completes, and the next activity will be started, algorithm begin to identify validity of all possible paths which the next activity belongs to first, and then search all exceptional paths. The result provides semantic support for scheduling next activity.

Algorithm 3 SearchAllExceptionPath(G_{Ctrl}, v_0)-search all exceptional paths

```
Input: G_Ctrl and any node $v_0$ in G_Ctrl,$v_0\neq n_e$
Output: all exceptional path $Path=\{p_m,...,p_n\}$ between $v_0$ and $n_e$
P = AllPath(G_Ctrl, $v_0$);
While P$\neq\emptyset$ do
        P $\leftarrow$ P - {$\forall P_i | P_i \in P$ };
        If ExceptionChecking ($P_i$) = FALSE then
                Path $\leftarrow$ Path $\cup$ { Pi };
return Path;
```

Through semantic supplement, search algorithm for exceptional path is actually a strategy of searching forward. The algorithm not only offers more semantic support to schedule activities in workflow, it but also provides decision support to workflow participants, and avoids executing unsuccessful paths.

3.3 Algorithm Analysis

When identifying exceptional path, the set of data dependency rules applied in some paths are conditions, so we consider the time complexity of traversing paths as the performance of algorithm. In order to search all paths from node v_0 to n_e, algorithm 1 must re-expand those visited nodes. Fully expanded graph is equivalent to a tree whose leaf-nodes are v_0, and therefore, the time complexity of the algorithm depends on two aspects. The first is the amount of nodes whose out-degree>1. These nodes determine the amount of re-expanded nodes. The second is the position of nodes whose out-degree>1. The closer to v_0 these nodes are, the more amount of successors of v_0 the algorithm needs to expand. Assume that *OD(v)* is out-degrees of any node *v* from v_0 to n_e and *L* is the longest path from v_0 to n_e, where *L*>1, then the amount of paths is $OD(v_0)+\sum(OD(v_i)-1)$, where $OD(v_i)>1$. The time complexity of search algorithm employing adjacent table is $O(L*(OD(v_0)+\sum(OD(v_i)-1))$. In the worst situation, all nodes whose out-degree>1 are close to v_0, the longest path is near to *L*. In the best situation, all nodes whose out-degree>1 are close to v_e, the longest path is far less than *L*. Therefore the actual time complexity of traveling graph is between $O(OD(v_0)+\sum(OD(v_i)-1))$ and $\leq O(L*(OD(v_0)+\sum(OD(v_i)-1)))$.

The soundness and reachability of workflow schema (graph) can be guaranteed by Ref.[12], so the algorithm 1 can search all paths from v_0 to n_e. Because the intention applying data dependency rules is to detect exceptional path, it is beneficial to a WfMS if those exceptional paths are identified earlier. Only when *Available* $(n_i.D)=TRUE$ is met, can data dependency rules related with $n_i.D$ be employed.

4. Maintenance of Data Dependency Rule

Data dependency between activities is abstracted from business rules. Nowadays business rules are mined mainly through (1) gaining experience of enterprise management; (2) data mining technology; (3) model mining technology aimed at workflow [12]. In fact, *WfMC* does not define data dependency rule [11], and only define the transfer condition and workflow relevant data. The former is equivalent to control rule, while the latter is data mainly used in control rule. We have expanded the definitions of *WfMC* above.

Data dependency rule can be converted to control rule directly. This changes the structure of workflow schema. The conversion, however, depends on many factors, such as traditional business custom of enterprise, soundness of workflow model, and data independence between activities. Our method is based on changeless model structure of workflow in this paper. We use algorithm instead of conversion to realize detection of exceptional. The conversion is a research issue in future.

5. Examples

We give an example of equipment maintenance process in most manufacture enterprises. The following is an explanation of business process. A request for repair is submitted to ED(Equipment Department) when a equipment failure occurs. If it is a common failure, repairers of workshop can fix it, or another request is submitted to ED again. At this time, engineers from ED are dispatched to handle the failure. The repair steps are composed of evaluating failure, dispatching, picking and repairing. According to the engineer, if the failed spare part must be repaired on commission, a third request is submitted to ED for repairing on commission. The meanings of business activities are labeled on the nodes in Fig.3, and the words near the arcs are conditions, representing control dependency rules between activities. In order to discuss expediently, the workflow schema is unwrapped, in which node X,X′and X″denote the same activities, roles of activities are ignored. To satisfy definition 1, let's construct a virtual end activity n9. Table.1 shows consumed time of each activity, involving manually and automatically executed by computer.

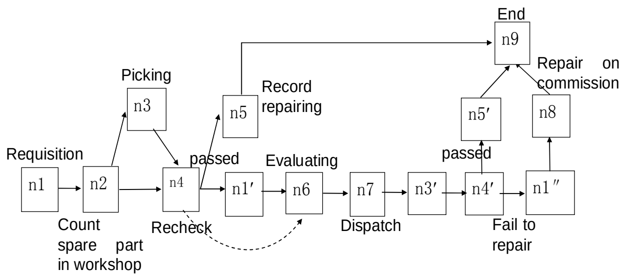

Fig. 3 Equipment maintenance process

Let us consider the data dependency rule IsNotConsign(n4.PartID)→n6.PartID. Corresponding explanation is to judge if PartID needs repairing on commission when activity n4 completes, and the result will have an impact on activity n6. In fig.3, if dotted line is considered, the graph is G_{Data}, or the graph is G_{Ctrl}. Let IsNotConsign(n4.PartID)=FALSE, i.e. PartID must be repaired on commission. After activity n1′completes, Algorithm 3 is executed. The following is computing steps.

(i) By AllPath(G_{Ctrl}, n1′), search all paths between n1′and n9, the result is PATH ={ Path1, Path2}, see the first column in Table 2. Note that n1′ and n1″ are the same activity.

(ii) For every $Path_i$ in PATH, evaluate ExceptionChecking ($Path_i$). The computing process is showed in the column 2, 3 and 4 of Table.2

Table 1: Executing Time of Activities (A:activity, ET:Exection time)

A	ET(mins)	A	ET(mins)	A	ET(mins)
n1	2	n4	5	n7	2
n2	0.1	n5	5	n8	2
n3	1	n6	3	n9	0

Table.2 Computing process (P: IsNotConsign(n4.PartID),F:FALSE)

All paths started by n1′	P	Impacted activities by P	Available (n4. PartID)	Detecting result	ET
Path1=< n1′,n6,n7, n3′,n4′,n5′,n9>	F	n6	TRUE	exceptional	18
Path2=<n1″,n8,n9>	F	none	TRUE	normal	4

Analyzing the computation result:

(i) It can be seen from the computing result that after n1′executes, Path1 should be avoided and turned to Path2 to continue execution. Not only executing time can be shortened, but also the resource consumed by Path1 can be cut.

(ii) Data dependency rule IsNotConsign can be converted to control rule. However the conversion will change workflow model, which violates our approach, as mentioned in section 1. Our approach has the same effect as that of changing workflow model.

(iii) How to apply the searching algorithm in a workflow also depends partly on rationality of practical business process. In the example above, if searching algorithm executes before node n1′, it will get the same result, i.e. repair on commission as soon as workshop checks repairing (n4), not submitting request to ED(n1′). However the two business activities n4 and n8 can not form rational logic, and do not accord with business custom.

6. Conclusions and Future Work

By analyzing workflow foundational feature, when routing workflow, data dependency rule can be considered as an important supplement and restrictive rules to search and identify exceptional paths of business process beforehand. This approach not only avoids complexity modeling caused by new rules and additional cost, but also keeps rationality of business process. In addition, detecting exceptional paths also saves resources, especially those one-off resources.

Although searching algorithm of graph can solve exceptional detection about semantic failure, for a complex workflow schema, searching process is time-consuming. In the future, the algorithm needs to be enhanced to reduce complexity of searching space, so that the approach can deal with more complex workflow model.

Acknowledgments

The Project is supported by the National Natural Science Foundation of China No. 60573086; The National High-Tech. R&D Program for CIMS, China Grant 2003AA4Z3210, 2003AA413023 and 2002AA413310.

Reference

[1]Hollingsworth D. The workflow reference model, Document No. WfMC-TC-1003. Workflow Management Coalition.1995.

[2]C.Beckstein ,J.Klausner. A meta level architecture for workflow management. Society for Design and Process Science,1999,3. Volume 3,No.1,15-26.

[3]Zongwei Luo, Amit Sheth, Krys Kochut, et al. Exception Handling for Conflict Resolution in Cross-Organizational Workflows. University of Georgia, 2002.4.10.

[4]Michael Adams, David Edmond, and Arthur H.M. ter Hofstede. The application of activity theory to dynamic workflow adaptation issues. In Proceedings of the 2003 Pacific Asia Conference on Information Systems (PACIS 2003), Adelaide, Australia, 2003.7.1836–1852.

[5]M. Klein, C. Dellarocas, A knowledge-based approach to handling exceptions in workflow systems. Computer Supported Cooperative Work (CSCW) ,2000,9 (3/4):399–412.

[6] T. Murata, A. Borgida, Handling of irregularities in human centered systems: a unified framework for data and processes. IEEE Transactions on Software Engineering. 2000.10, 26 (10):959– 977.

[7] Zongwei Luo , Amit Sheth , Krys Kochut , John Miller, Exception Handling in Workflow Systems, Applied Intelligence, v.13 n.2, 9-11 2000: 125-147

[8] C. Hagen, G. AlonsoException Handling in Workflow Management Systems.IEEE Transactions on Software Engineering, Vol. 26, No. 10, 10 2000.

[9]Oberweis, A. Specification of techniques for handling exceptions with Petri nets. Automatisierungstechnik 1992.40(1) :21-30.

[10]FAN Yushun. Foundation of Workflow Management Technique. Beijing: Tsinghua University Press and Springer, 2001. (in Chinese)

[11]W.M.P van der Aalst. The application of Petri nets to workflow management. Journal of Circuits,Systems and Computers, 1998,8(1):21-66.

[12]W.M.P.Van der Aalst. Verification of Workflow Nets. In P.Azema and G.Balbo, Editors, Application and Theory of Petri Nets 1997, Volume 1248 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1997.407-426.

LI Hai-bo received the B.S. and M.S. degrees from Heilongjiang University in 1994 and Northeast Agricultural University in 2001, respectively. He is presently a Ph.D candidate at Center of Intelligent Computing of Enterprises, School of Computer Science and Technology in Harbin Industrial of Technology of China. His current research areas include ERP, workflow system and component-oriented development.

ZHAN De-chen He is He is a doctor, professor and doctoral supervisor of HIT, His research interest includes modern enterprise management, data and knowledge engineering , software reconstruction and reuse

XU Xiao-fei, He is He is a doctor, professor and doctoral supervisor of HIT, His research interest includes enterprise intelligent computing, management and decision information system, ERP, supply chain management, E-commerce and business intelligent, knowledge engineering and application.