http://www.zju.edu.cn/jzus; E-mail: jzus@zju.edu.cn
ISSN 1009-3095 *Journal of Zhejiang University SCIENCE* V.4, No.6, P.643-650, Nov.-Dec., 2003

Approach for workflow modeling using π-calculus

YANG Dong (杨 东)[†1], ZHANG Shen-sheng (张申生)[2]

([1] *Department of Industrial Engineering, Shanghai Jiaotong University, Shanghai 200030, China*)
([2] *Department of Computer, Shanghai Jiaotong University, Shanghai 200030, China*)

[†]E-mail: dongyangcn@hotmail.com

Received Sept.28, 2002; revision accepted Apr.29, 2003

Abstract: As a variant of process algebra, π-calculus can describe the interactions between evolving processes. By modeling activity as a process interacting with other processes through ports, this paper presents a new approach: representing workflow models using π-calculus. As a result, the model can characterize the dynamic behaviors of the workflow process in terms of the LTS (Labeled Transition Semantics) semantics of π-calculus. The main advantage of the workflow model's formal semantic is that it allows for verification of the model's properties, such as deadlock-free and normal termination. Moreover, the equivalence of workflow models can be checked through weak bisimulation theorem in the π-calculus, thus facilitating the optimization of business processes.

Key words: Workflow modeling, π-calculus, Business process modeling
Document code: A **CLC number:** TP311.5

INTRODUCTION

With constant changing markets and environments, it is an urgent task for enterprises to enhance its agility and flexibility (Zhang, 1996). By optimizing internal business processes and organization structure, BPR (business process reengineering) enables enterprises to adapt to changing environment, thus improving its agility. Workflow management technology is an important infrastructure for implementing BPR. Workflow is defined as part or total automation of business processes by routing task, information, and documents, between participants according to business rules, in order to achieve the business goals (Lawrence, 1997). Workflow modeling is computerized representation of business processes. The objective of workflow modeling is to provide high-level specification of business processes. When workflow modeling involving a number of concurrency and alternative constraints becomes complex, error can be introduced into the model. Such modeling inconsistency may lead to undesirable execution of some or all instances of a workflow. It is of vital importance that a workflow model be properly defined, analyzed, and verified before being deployed in a workflow management system. However, most workflow products do not give any support for the verification of workflow models. In contrast to other modeling approaches, such as graphical and semi-formal ones, the primary advantage of the formal modeling approach is to allow for verification. With rigorous mathematical foundation and analytical tools, Petri net based approach is thought of as main formal method for modeling and verifying workflow models. However, the main problem with Petri net is that it does not posses a complete algebra of operation. For example, there are no suitable methods for composing Petri nets to obtain larger Petri nets by using concurrency operator. The compositionality of the formal method is especially important for workflow verification. When the model becomes more and more complex, the difficulties with the verification of model will increase. Verification of a large model may become computationally infeasible. By verifying parts of the specification forming the whole workflow model, and then using a black box for representation of it for subsequent verification, this compositionality of formal methods reduces the complexity of verification tasks. π-calculus is a kind of computing model for representing concur-

rent systems and can express the interactions between evolving processes (Milner, 1999). As a variant of process algebra, π-calculus provides a way for constructing high-level system by composing its sub-systems using concurrency operator. The compositionality is one of the major advantages in π-calculus and thus will reduce the complexity of verification of larger workflow model. In this paper, we present a new approach for modeling workflow based on π-calculus. Another advantage in π-calculus is that it enables the checking of the equivalence between two processes in term of bisimulation theorem. As a result, workflow models can be checked for equivalence, which is especially useful in business processes optimization requiring one process to be substituted for another without changing business functions.

This paper is organized as follows. Section 2 gives a background of workflow modeling. In Section 3, we introduce the syntax and semantics of π-calculus. Section 4 describes our approach for modeling workflow based on π-calculus. Section 5 gives an example to illustrate the formalism. Section 6 addresses the analysis and verification of π-calculus based workflow model. In Section 7, related works are compared. Finally, in Section 8 we point out future work.

WORKFLOW MODELING APPROACH

Workflow normally consists of activities. For example, in an order processing workflow, typical activities include the input of customer order into computer, the approval of order, and the check of stock, etc. There exists complex dependency among activities, such as concurrency, non-determination decision, etc. Workflow modeling can be divided into a part for representing activities, including the assignment of the resource to activities, and one for describing dependency among activities.

1. Activity

Activity is an atomic unit of work that cannot be divided further. According to the type of triggering, activity can be classified into four groups: automatic activity, manual activity, message-triggering activity and timing-triggering activity. Automatic activities do not require human intervention, and once triggered, will start execution. Manual activities require interaction with humans. The enabled activity can only be executed when users choose the work item from user's worklist. The message-trigger activity is triggered by the arrival of message or event, such as email. Timing-triggering activity is triggered by timer. The activities must be started at some time point or during time intervals. For example, the sum of one-day's sales is calculated in 24: 00 everyday. The processing of orders is done from 8:00 AM to 5:00 PM everyday.

2. Control dependency

There are complex dependencies between activities in workflow processes. In order to represent more advanced dependencies among activities, WFMC (Workflow Management Coalition) defines basic control elements, which include AND-split, AND-join, OR-split, OR-join (Lawrence, 1997). The AND-split node is used to split single control into concurrent control threads. And the AND-join is employed to synchronize concurrent branches and converge them into single control thread. The OR-split node represents the case where only one branch is chosen when there exist alternative branches. Moreover, there is a special OR-split, which chooses alternative braches based on explicit condition, called condition dependency. The OR-join node denotes that multiple branches are converged to one control thread without synchronization. The Petri net semantics of the control elements are shown in Fig. 1 (Van der Aalst, 1996).

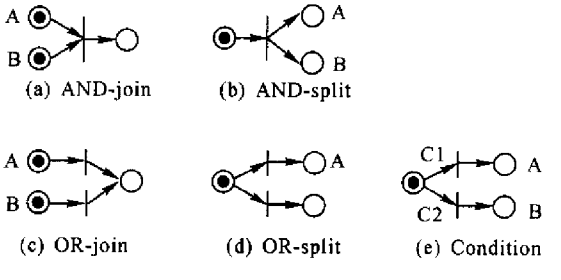

Fig.1 Petri net representation of dependencies

SYNTAX OF π-CALCULUS

π-calculus comes from CCS (Calculus of

Communicating Systems), representing the interactions between processes. In π-calculus, there are two basic concepts. One is name. The channels (ports), variables, data are names. Different from CCS, π-calculus does not distinguish between them. The other is processes (agents), representing the entity in systems. The interaction between processes is done through a pair of complementary ports. Polyadic π-calculus is an extended π-calculus allowing the sending or receiving of more than one name through ports (Milner, 1989). The syntax of processes expression in polyadic π-calculus is given below. The lowercase letters denote names (such as x, y) while uppercase letters denote processes.

(1) Input prefix $a(\boldsymbol{x}).P$: The process first receives a name on port a, then behave as P.

(2) Output prefix $\bar{a}<\boldsymbol{x}>.P$: The process first outputs the name x on a, then behave as P.

(3) Summation $P_1 + P_2$: The process behaves like eitheror P_1 or P_2.

(4) Composition $P_1 \mid P_2$: is a process that can do anything that P_1 or P_2 can do, and moreover communication between P_1 and P_2 can occur if one process outputs a name and the other inputs a name on the same port.

(5) Restriction $(vy)P$: is a process that behaves like P, but the name y cannot be used for communicating with the environments.

(6) Matching $[x = y].P$: is a process which behaves like P if x and y are the same name. Otherwise, it can do nothing.

(7) A defined agent, $A(y_1, \cdots, y_n)$, has a unique defining equation, $A(x_1, \cdots, x_n) \stackrel{\text{def}}{=\!=} P$ where $x_1, \cdots, x_n$ occur free in P. The process $A(y_1, \cdots, y_n)$ behaves like $P\{y_1/x_1, \cdots, y_n/x_n\}$.

Moreover, the expression $\{\boldsymbol{a}/\boldsymbol{b}\}P$ in the π-calculus represents the substitution of $\boldsymbol{a}$ for $\boldsymbol{b}$ in P. The formal operational semantics of the processes expression is defined and explained in (Milner *et al*., 1992). For example,

$$(\cdots + y<\boldsymbol{x}>.P) \mid (\cdots + y(\boldsymbol{z}).Q) \xrightarrow{\tau} P \mid Q\{\boldsymbol{x}/\boldsymbol{z}\}$$

The above rule shows that two processes can communication (handshake) via complementary ports, resulting in silent action τ. And after this communication, all free occurrence of z in Q are replaced by x.

METHODS FOR MODDELING WORKFLOW USING π-CALCULUS

As a variant of process algebra, π-calculus offers a way for constructing a system through composing subsystems. Workflow can be regarded as a network of activities interacting with each other. Therefore, we can model activity as a process in π-calculus. According to the modeling approach mentioned in Section 2, we first introduce the representation of activity using process expression in π-calculus. Then the representations of dependencies are presented.

1. The specification of activity

Activity is the central concept in workflow and is atomic work unit that cannot be divided further. Resources are needed to perform activity. Typical resources include human users, applications, tools, etc. Resources are used during the execution periods of activity.

From the viewpoint of workflow management systems (WFMS), workflow is a means for coordinating activities, but their execution is outside workflow management. WFMS is only concerned with the interaction of activities, i.e. the management of activities. For example, when an activity is enabled, WFMS has to assign resource to perform the activity. After activity is completed, WFMS is notified of the completion and then WFMS has to release the assigned resource. Thus, activity acts as a "black-box" that interacts with the external environment though ports. We can use processes in the π-calculus to represent activities. And we introduce a special action name done, to denote the termination of activity.

Definition 1 Activity can be represented as process in π-calculus.

Definition 2 If activity A is automatic activity, then its corresponding representation of process is:

$$A =_{def} start.\overline{req\_resource}<case\_id, activity\_id, resource\_id, assigned\_resource>.assigned\_resource(case\_id, activity\_id, resource).\bar{a}.finish.\overline{rel\_resource}<resource>.\overline{done}$$

Where uppercase name of activity (such as A) denotes processes and lowercase name of activity (such as a) denotes the execution of activity.

The above definition shows that the process requests resource through port *req-resource* after activity is enabled. The port has input names, i.e. *case_id*, *activity_id*, *resource_id*, *assigned_resource*, which represent the identity of workflow instances, the identity of activity, the identity of resource required, and private channel respectively. The private channel *assigned_resource* is used for the receiving of assigned resource. The input name *resource* of the port *assigned_resource* represents the resource to be assigned. After the resource has been assigned, the process executes activity a by interacting with external application or tool. When the activity has completed, the process gets the acknowledgement of completion of activity through port *finish*. Then process releases resource through port *rel-resource*. Finally, the process outputs the message of activity termination, and fire the next activity.

Definition 3 If activity A is a manual activity, the corresponding process can be defined:

$$A =_{def} start.\ \overline{req\_resource} < case\_id,\ activity\_id,\ resource\_id,\ assigned\_resource > .\ assigned\_resource(case\_id,\ activity\_id,\ resource).\ user().\ \bar{a}.\ finish.\ \overline{rel\_resource} < resource > .\ \overline{done}$$

Definition 3 is similar to definition 2, except that the process will wait for the message of user's triggering activity through port *user* after resource is assigned.

Definition 4 If activity A is a message-triggering activity, the corresponding process can be defined:

$$A =_{def} start.\ event(eventID).\ \overline{req\_resource} < case\_id,\ activity\_id,\ resource\_id,\ assigned\_resource > .\ assigned\_resource(case\_id,\ activity\_id,\ resource).\ \bar{a}.\ finish.\ \overline{rel\_resource} < resource > .\ \overline{done}$$

Definition 4 is similar to definition 2, except that the process will wait for the arrival of event through port *event* after the activity is enabled. The port *event* has one input name *eventID* which is the identity of event.

Definition 5 If activity A is a time-triggering activity, the corresponding process can be defined:

$$A =_{def} start.\ \overline{timer} < fromtime,\ totime,\ t > .\ t().\ \overline{req\_resource} < case\_id,\ activity\_id,\ resource\_id,\ assigned\_resource > .\ assigned\_resource(case\_id,\ activity\_id,\ resource).\ \bar{a}.\ finish.\ \overline{rel\_resource} < resource > .\ \overline{done}$$

Definition 5 is similar to definition 2, except that the process will inquire the time from timer through port *timer* after the activity is enabled. The port *timer* has output names *fromtime*, *tomine*, which represent the beginning and end of the period of activity's execution, as well as private port t respectively. If *totime* is NULL, this represents that activity will be started from some timepoint.

The private port t is later used for the receiving of message of trigger by timer.

2. The specification of dependency

As mentioned in Section 2, dependency among activities contains AND-join, AND-split, OR-join, OR-split and Condition. In the following definitions, the process expressions of activity A, B, C and D, i.e. A, B, C, D, can be defined in terms of definition 1 – 5 and will be omitted for simplicity.

Definition 6 If activity A and activity B have sequence dependency and their corresponding processes expression is A, B respectively, then the sequence dependency can be defined as:

$$A\ Before\ B =_{def} (vp)(\{p/done\}A \mid \{p/start\}B)$$

It shows that after the process A has complete activity the process A will communication with the process B through complementary ports (p, p), and then process B will be executed.

Definition 7 If activity A and activity B have OR-split dependency (Fig. 1d) and their corresponding processes expression is A, B respectively, then the OR-split dependency can be defined as:

$$A\ OR\text{-}split\ B =_{def} (vp)((start.\ p) \mid (\{p/start\}A + \{p/start\}B))$$

It can be seen that after the control element OR-split is enabled (through port *start*), then communication with process A or process B

through complementary ports $(\bar{p}, p)$ happens, and thus A or B will be executed.

Definition 8 If activity A and activity B have OR-join dependency (Fig. 1c) and their corresponding processes expression is A, B respectively, then the OR-join dependency can be defined as:

$$A \ OR\text{-}join \ B =_{def} (\nu q)((\{q/done\}A + \{q/done\}B) \mid q.\overline{done})$$

It shows that after process A or process B has finished activity, it will trigger the execution of action done, thus firing the next process.

Definition 9 If activity A and activity B have AND-split dependency (Fig. 1b) and their corresponding processes expression is A, B respectively, then the AND-split dependency can be defined as:

$$A \ AND\text{-}split \ B =_{def} (\nu p)(start.(p \mid p) \mid (\{p/start\}A \mid \{p/start\}B))$$

It shows that the process A and B both will be started concurrently after control element AND-split is enabled. And private port p is used to fire the processes A and B.

Definition 10 If activity A and activity B have AND-join dependency (Fig. 1a) and their corresponding processes expression is A, B respectively, then the AND-join dependency can be defined as:

$$A \ AND\text{-}join \ B =_{def} (\nu p)((\{q/done\}A \mid \{q/done\}B) \mid (\nu w)(q.w \mid q.w \mid q.w \mid w.w.\overline{done}))$$

In the above definition, the process A, B will respectively communicate via port pairs $(q, \bar{q})$ and the private channel is used to synchronize the two processes.

For the same reason, it is not difficult to describe the specification of dependencies where more than two activities are connected by AND-split (join), OR-split(join).

Definition 11 If the condition expression ϕ is true, then activity A will be executed; otherwise activity B will be started. We can use the following process expression to define it.

$$\text{if } \phi \text{ then } A \text{ else } B =_{def} (\nu \ xd_1 d_2) \ start.\overline{condition} < x, \phi > .x(result).([result = True]\bar{d}_1 + [result = False]\bar{d}_2)) \mid (\{d_1/start\}A \mid \{d_2/start\}B)$$

In the above definition, the port *condition* outputs a private name x and the condition expression ϕ to be evaluated. The private port x is later used to receive the value of condition ϕ, according to which the process A or process B is executed.

3. The specification of subprocess

In workflow modeling, it is necessary to provide modular construct (such as subprocess) in order to reuse part of the workflow specification and support modeling at different level of abstraction. And subprocess is composed of activities, or other subprocesses.

Definition 12 If A is a subprocess whose corresponding process expression in the π-calculus is P_{sub} and its parent process is B, the parent-subprocess dependency can be defined as:

$$A \ IS\text{-}SUB\text{-}OF \ B =_{def} (\nu \ notify \ ack)(start.notify.ack.done \mid \{notify/start, ack/done\}P_{sub})$$

The definition shows that the parent process B sends the notification of triggering subprocess through port *notify* after it receives the message of start from other processes. Thus the subprocess is fired and then is executed. After the subprocess finishes, it sends acknowledge message *ack* to its parent process and as a result, the parent process can indicate its completion by sending message *done* to other processes.

EXAMPLE PROBLEM

In order to illustrate how to describe workflow models using π-calculus, this section will give an example of travel reservation. Fig. 2 is a workflow process for travel reservation. The workflow process runs as follows: Once customer requests reservation, the desktop select and book trip for the customer. If customer does not feel satisfied with it, the travel reservation will be given up and the process terminate. Otherwise, sending acknowledgement to customer and making payment can be started concurrently. Once the two activities have finished, the desktop prepares document and then sends document to the customer. And the workflow process finishes.

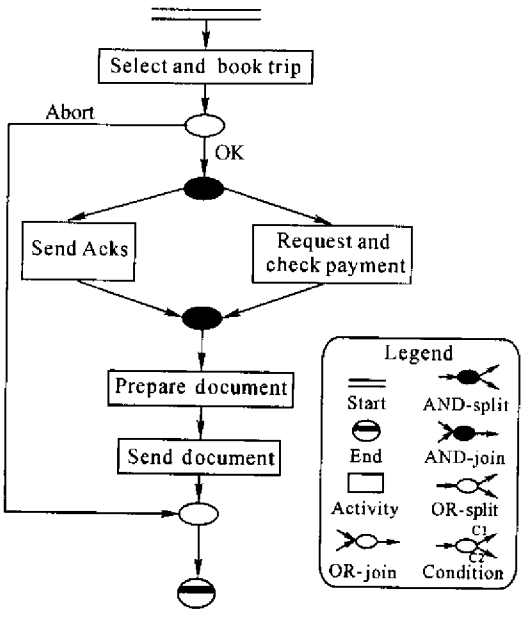

Fig.2 Workflow for travel reservation

The process expression which the activity "select and book trip" corresponds to is defined according to definition 2.

$SELECETTRIP =_{def} start . \overline{req\_resource}(case\_id, activity\_id, resource\_id, assigned\_resource) . assigned\_resource(case\_id, activity\_id, resource) . \overline{selecttrip} . finish . \overline{rel\_resource}(resource) . \overline{done}$

For the same reason, the specification of the activities "Send Acks", "Request and check payment", "Prepare document", "Send document" can be described. Due to space limitation, we choose not to list here. Furthermore, we define a special process named "Terminate" that represents the termination of the workflow process.

$TERMINATE =_{def} start . \overline{done}$

The process "Terminate" does not assign resource and also need not to execute activity. It acts as an indication of the termination of workflow processes.

Therefore, the specification of workflow process about travel reservation can be defined:

$TRAVELRESERVATION =_{def} SELECTTRIP$ **Before**

if ϕ **then** ($SENDACK$ **Parallel** $REQUEST\text{-}CHECK$)

Before ($PREPAREDOC$ **Before** ($SEND\text{-}DOC$ **Before** $TERMINATE$))

)

else $RERMINATE$)

In the above specification, we introduce another dependency between activities, i.e. *Parallel*. The *parallel* dependency consists of AND-split and AND-join. For example, the activity "Send Acks" and the activity "Request and check Payment" that are connected by AND-split and AND-join form *Parallel* dependency. We can define *Parallel* dependency based on the semantics of AND-split and AND-join.

$A\ Parallel\ B =_{def} (v\ pq)((start . (\bar{p} \mid \bar{p}) \mid (\{p/start\}\{q/done\}B \mid \{p/start\}\{q/done\}C)) \mid (vw)(q . w \mid q . w \mid w . w . \overline{done}))$

The whole workflow process was triggered by an external event, i.e. *EventCustomerReq*.

$EventCustomerReq = \overline{start} . 0$

Thus, workflow process can be described as follows:

$(v\ start)(EventCustomerReq \mid TRAVELRESERVATION)$

MODEL ANALYIS AND VERIFICATION

The aim of formalizing workflow model is to enable the analysis and verification of the workflow model. Based on the process semantics in the π-calculus, we can check whether two modes are equivalent or whether the model satisfies correct requirement, such as deadlock-free properties. When analyzing the properties of workflow model, the coordination between activities of workflow is a major factor to be considered, and the details about how WFMS manages resources have no impact on the result of analysis on models. Thus, we can omit the representation of details about activity, such as assignment of resource as well as type of activity. Thus, define 2 to define 5 can be described simply as:

$$A =_{def} start . a\ \overline{done}$$

1. Correctness properties

In order to ensure the correctness of the workflow model, the process models must satisfy two basic requirements (Van der Aalst, 1998): (1) Starting form initial state, final state will always be reachable. This means that workflow

processes must normally terminate. When workflow process is complete, there are no dangling activities. (2) There does not exist deadlock. This means that any activities can be executed. Thus, we use mu-calculus to describe the correctness requirements (Bradfield and Stirling, 2001).

correctness requirements

Requirement 1 If G is a workflow model whose process expression in the π-calculus is P_G, then P_G has to satisfy:

$$P_G \models \mu Z . < finaltask > \boldsymbol{tt} \vee (< - > \boldsymbol{tt} \wedge [-] Z)$$

where finaltask denotes the activities that the final node in workflow model represents.

Requirement 2 If G is a workflow graph whose process expression in the π-calculus is P_G, then P_G has to satisfy:

$$P_G \models \nu Z . < - > \boldsymbol{tt} \wedge [-] Z$$

Requirement 1 says that workflow process normally terminate. And requirement 2 shows that the model is deadlock-free.

2. Model equivalence

It is often useful to transform one model into another equivalent model when modeling workflow or optimizing business processes. Based on weak bisimulation in the π-calculus, we can check whether two process models can have the same behaviors.

Definition 13 $P \approx Q$ iff, for all $\alpha \in ACT$

(1) whenever $P \xrightarrow{\alpha} P'$, then $\exists\, Q', Q \xrightarrow{\hat{\alpha}} Q'$ and $P' \approx Q'$.

(2) whenever $Q \xrightarrow{\alpha} Q'$, then $\exists\, P', P \xrightarrow{\hat{\alpha}} P'$ and $P' \approx Q'$.

Here "$\approx$" is a weak equivalence between processes.

Definition 14 If processes expressions of two workflow models are denoted by A and B respectively, then the two models are identified as equivalent iff $A \approx B$.

Definition 15 If processes expressions of workflow models is denoted by A, then there exists deadlocks iff

$\exists\, t, A \xrightarrow{t} A'$ and $A' \sim 0$.

Where t is a sequence of actions and "$\sim$" is strong equivalence.

The π-calculus analytical tool MWB (Mobility Workbench) not only checks the equivalence of two processes, but also verifies whether deadlock exits in the processes (Victor and Moller, 1994). Furthermore, MWB provides interactive simulation to observe the dynamic behaviors of process. Thus, we can automatically check workflow models with the help of existing π-calculus analytical tool.

RELATED WORK

Much research effort is being carried out in the workflow modeling field, yielding a number of approaches to workflow processes representation. These efforts range from the use of graphical modeling approach, to rule-based languages, and formal languages. In FlowMark, graphical modeling method is adopted and workflow model is represented by a directed diagram where node denotes tasks and arc between nodes depicts control flow or data flow of tasks (Alonso and Mohan, 1997). The WIDE project also uses a graphical language and then the graphical model is converted into internal representation in term of WFDL (Wide workflow Description Language) (Casati *et al.*, 1997). A main disadvantage of graphical modeling language is that workflow model suffers from the lack of rigorous well-defined semantics, making it impossible to reason and analyze the workflow model. Rule-based formalism, as an executable approach, especially Event-Condition-Rule rule, can be used to represent workflow model. In TriGSflow workflow system , the ordering of activities was described by ECA rules bound to objects (Kappel *et al.*, 1998). Also, rules similar to ECA are used to depict state dependencies and value dependencies between activities in METEOR project (Krishnakumar and Sheth, 1995). Despite the flexibility in dealing with dynamic modification of model, rule-based approach has to depend on other formal method, such as Petrinet and temporal logic, to represent the model semantics. Therefore, it is extremely difficult to analyze and reason rule-based workflow models. The main advantage of formal approach is that it enables the analysis and verification of the workflow model. At present, Petri net is mainly used to model workflow due to its rigorous mathematical foundation (Van der Aalst, 1998; Ellis and

Nutt, 1993; Bandinelli *et al.*, 1994). In the SPADE project, SLANG language, an extension of high-level Petri net, was adopted to model software process and support the defining, enacting, customizing of process models (Bandinelli *et al.*, 1993; 1994). In contrast, our π-calculus based approach not only enables the formal representation of the workflow model, but also characterizes how WFMS manages activities, such as assignment and release of resource. With respect to model analysis, one of its main advantages over the Petri net based approach is the compositionality, making composite verification possible and thus reducing the complexity of verification task. Moreover, the checking of model equivalence is not yet addressed by Petri net based approach.

CONCLUSIONS

By describing workflow specification using π-calculus, the workflow model can express the dynamic behavior of business processes. With rigorous process semantics, the model can be verified whether it satisfies certain properties, such as correctness requirement. Moreover, according to bisimulation theorem in π-calculus, the model can be checked for equivalence. Although the main disadvantage of π-calculus is that it is difficult for most people to understand it very well, the π-calculus can be regarded as an internal representation of workflow model for analysis and verification. When combined with graphical representation for user, our approach of formalizing workflow modeling provides an alternative way for workflow modeling and verification.

References

Alonso, G. and Mohan, C., 1997. Workflow Management: The Next Generation of Distributed Processing Tools. *In*: S. Jajodia, L. Kerschberg (Eds.), Advanced Transaction Models and Architectures, Kluwer Academic Publishers.

Bandinelli, S., Fuggetta, A. and Ghezzi, C., 1993. Software process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, **19**(12): 1128 – 1144.

Bandinelli, S., Fuggetta, A., Ghezzi, C. and Lavazza, L., 1994. SPADE: An Environment for Software Process Analysis, Design, and Enactmen. *In*: Finkelstein, A., Kramer, J., Nuseibeh, B., editors. Software Process Modelling and Technology. John Wiley & Sons, London, England, p.223 – 244.

Bradfield, C. and Stirling, C., 2001. Modal Logics and Mu-calculi: An Introduction. *In*: Bergstra, A., Ponse, A., Smolka, S. A., editors, Handbook of Process Algebra. Elsevier Science, p.293 – 330.

Casati, F., Grefen, P. and Sanchez, G. ,1997. WIDE - A Distributed Architecture for Workflow Management. *In*: Proceedings 7th International Workshop on Research Issues in Data Engineering , Birmingham, England, P. 76 – 79.

Ellis, C. and Nutt, G., 1993. Modeling and Enactment of Workflow Systems. *In*: M. Ajmone Marsan, editor, Application and Theory of Petri Nets (LNCS691), Springer-Verlag, Berlin, Heidelberg, p.1 – 16.

Kappel, G., Rausch-Schott, S. and Retschitzegger, W., 1998. Coordination in Workflow Management Systems - A Rule-based Approach. *In*: Conen, W., Neumann, G., editors, Coordination Technology for Collaborative Applications – Organizations, Processes, and Agents (LNCS 1364), Springer, Berlin, Heidelberg, p.99 – 120.

Krishnakumar, N. and Sheth, A., 1995. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, **3**(2). p. 155 – 186.

Lawrence, P. (editor), 1997. Workflow Handbook 1997. John Wiley and Sons, New York.

Milner, R., 1989. Communication and Concurrency. Prentice Hall, New York.

Milner, R., Parrow, J. and Walker, D., 1992. A calculus for mobile processes, parts I and II. *Journal of Information and Computation*, **100**: 1 – 77.

Milner, R., 1999. Communicating and Mobile Systems: the π-Calculus. Cambridge University Press.

Van der Aalst, W.M.P., 1996. Petri-net-based Workflow Management Software. *In*: A. Sheth, editor, Proceedings of the NFS Workshop on Workflow and Process Automation in Information Systems, Athens, Georgia, p. 114 – 118.

Van der Aalst, W.M.P., 1998. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, **8**(1): 21 – 66.

Victor, B. and Moller, F., 1994. The Mobility Workbench-A Tool for the π-calculus. *In*: Dill, D., ed., Proceedings of the Conference on Computer-Aided Verification (CAV'94)(LNCS818), Springer-Verlag, Berlin, Heidelberg, p.428 – 440.

Zhang, S.S., 1996. From CIMS to dynamic alliance. *China Mechanical Engineering*, **7**(3): 17 – 22.