http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (5): 750-756, 2009
ISSN 1812-5638

Modeling and Design for Dynamic Workflows Based on Flexible Activities

[1]Peng Li and [1,2]Yuyue Du
[1]College of Information Science and Engineering,
Shandong University of Science and Technology, Qingdao 266510, China
[2]The State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing 100080, China

Abstract: A rapidly changing environment forces the workflow management systems to change their workflow processes ever more frequently. In order to improve the flexibility of workflow management systems, a dynamic workflow model is proposed in this study. The concepts of flexible activities and historical execution information are put forward to construct dynamic workflow models in this method. Each flexible activity is used to encapsulate a group of indeterminate factors, e.g., the constraint rules and optional sub-activities when reifying a flexible activity. Historical execution information is the executive logging of a previous workflow instance. Two algorithms are put forward to guarantee the correctness of sub-workflows and the global control of dynamic processes. Furthermore, a simple example is given to validate the proposed dynamic workflow model. However, this method cannot perform well if there existing loop structures in sub-workflows and the reifying processes of flexible activities are not intelligent enough.

Key words: Dynamic workflow, flexible activity, workflow model

INTRODUCTION

Workflow is a kind of business processes automated in whole or part and the documents, information or tasks are passed from one participant to another, according to a set of procedure rules (Van der Aalst and Basten, 2002). Workflow management technology aims at the automated support and coordination of business processes to reduce cost and increase efficiency. Many of the environments where workflow management systems (WFMS) are used are constantly changing, because, e.g., new customer's requirements have to be met, business processes are reengineered, or new laws demand to change the way business is carried out. These changes often require the workflow schema to be modified accordingly, by defining new workflow types or deleting/modifying existing ones. There are mainly two reasons for the constant changing of workflow models, (1) devisers lack enough knowledge while workflow models are designed, so as to lead to the procedures do not satisfy the actual needs and (2) something wrong or contingency have occurred in the implementation process. e.g., some mistakes happen due to the lapse of certain employee or new laws were promulgated (Van der Aalst and Jablonski, 2000).

Generally speaking, changes of workflows come from two layers, one is business model layer and another is business instance layer. Business model layer is the template of workflow when instantiating workflow instances. Business instance layer is the congregation of workflow instances. A modification of business instance layer only affects an individual case, but not affects the definition of a process model. In a medical treatment system, for example, individual instance must be adjusted based on an actual situation, since there are emergency cases which occurred frequently in hospitals. Changes from business model layer will affect all running workflow instances relying on this model and it is liable to cause the chaotic of workflow types' managements, as well as the definitions and executions come apart. Therefore, all related instances need to be adjusted for the modifications (Zhou *et al*., 2005). Workflows are divided into three groups on the basis of workflow modifications' characteristics:

- Flexible workflow
- Auto-adaptive workflow
- Dynamic workflow

Flexible workflow can make corresponding changes by means of the situation of model modifications and its difficulty is how to migrate the running workflow instances to the new schema.

Auto-adaptive workflow can handle the exceptions that occur in workflow instances implementation

Corresponding Author: Yuyue Du, College of Information Science and Engineering,
Shandong University of Science and Technology, Qingdao 266510, China

processes and its difficulty is how to tackle the unexpected exceptions.

Dynamic workflow can produce workflow instances in an uncompleted flow definition and it use a flexible activity to encapsulate a set of unsure factors. Its hard points are how to construct the sub models for flexible activities and how to guarantee the correctness of sub-workflows (Sadiq *et al*., 2001; Deng *et al*., 2004).

The issues of dynamic changes have been recognized by workflow communities for a long time and different approaches have been achieved so far (Rinderle *et al*., 2004). Van der Aalst and Basten (2002) introduced a concept of inheritance. This method provides a group of migration rules and preserves the inheritance relations between a new model and an old one, but if the new workflow model is not a super class or sub class of the old one, it is unable to complete the migration process. Agostini and DeMichelis (2000) proposed a method that assesses the correctness of migration by constructing the sequence model of workflows and their method only covers three kinds of modifications: parallelization, sequentialization and swapping. If other kinds of modifications occur, such as the modifications of a loop structure, adding or deleting certain activities, this method cannot work well. Yang and Wang (2008) proposed a migration method of workflow instances on the basis of the states of instances, where the consistency of historical execution information is checked and the modification information of the model as well as the states information of instances are used to realize the migration of workflow instances. This method is theoretically feasible, but when it is implemented in actual systems, the securing process of an equivalent state is quite troublesome and along with the enlargement of systems' scale, the computing complexity will increase promptly.

To enhance the descriptive ability for dynamic elements in workflows, some structures such as choice-merge and XOR-split are introduced in workflow models. All possible ways are included in model definitions, but this result in a procedure model that is too huge and difficult to understand. In addition, it is only practical when all possible situations are known, but it is not realistic in most cases. To avoid above limitations, the concepts Black Box and Pockets are introduced in some papers (Sadiq *et al*., 2001; Sun and Shi, 2003). Black box or pockets is used to encapsulate dynamic elements at modeling stage. The sizes of model definitions are reduced in these methods and they support dynamic workflows to some extent. However, it adds new elements into a workflow model and this measure increases the complexity of a model. Moreover, these studies did not discuss how to unfold black box or pockets. A dynamic workflow model based on ECA rules and activities composition was put forward by Deng *et al*. (2004). An adaptive activity was used to represent a set of unsure factors at modeling stage and an algorithm was proposed to check the rationality of a sub-process when finish reifying an adaptive activity. However, it did not present how a running workflow instance could get the information needed when reifying adaptive activities. Present study is investigated based on the above study and it mainly concentrates on flowing problems:

- How to fix the sub-process model under an uncompleted flow definition?
- How to guarantee the correctness of sub-workflows?
- How to make workflow instances carried out smoothly according to sub-workflow models?

MODEL OF DYNAMIC WORKFLOW

A dynamic workflow model with uncompleted definition is put forward in this section. There are two kinds of activities in this model: general activities and flexible activities. General activities represent the events that are stable and determinable and these activities are assured on the initialization of a workflow instance. Flexible activities represent special activities or sub-workflows and they cannot be completely defined beforehand.

Definition 1: A workflow is a five-tuple: W = <ID, M, Q, S, R>, where ID is the sole identifier of a workflow; M is the general information of a workflow and M = <Founder, Date, Version, Info> is a four-tuple, where four parameters represent the following information, respectively: the founder, the issue date, the model version and the descriptive information of an instance; Q = (N; F), where N is the set of all activities appearing in the workflow model, including general activities and flexible activities; F is the set of flow relations between activities, including the relations between general activities and also the relations between general activities and flexible activities and $F \subseteq N \times N$. S is the state of a workflow instance; R is the historical information of a previous workflow instance.

Every activity n_i of N is a four-tuple, n_i = <$Name_i$, $Type_i$, Cr_i, $State_i$>, where $Name_i$ is the name of activity n_i. $Type_i \in$ {General, Flexible} and it is the type of activity n_i; if $Type_i$ = General, Cr_i is null and $State_i \in$ {Notactivied, Actived, Running, Completed}; if $Type_i$ = Flexible, $State_i$ is null and Cr_i includes the rules being followed in the reifying process of flexible activity n_i. Cr_i = ($Event_i$, $Rule_i$, $Restrict_i$) is a three-tuple:

(1) $Event_i$ is a set of all activities being selected, while reifying flexible activity n_i, $Event_i = \{e_1, e_2, e_3 \ldots e_n\}$; the quantity of activities in $Event_i$ can increase or decrease dynamically according to external environments

(2) $Rule_i$ is a group of rules being followed while choosing activities in reifying process of flexible activity n_i. These rules restrict the consistency relations between activities. For example, some activities must be chosen or not; some activities must be selected simultaneously and some incompatibly. These rules can be changed based on the actual requirements dynamically and are formally described in the following ways:

- +e: Means that activity e must be selected
- −e: Means that activity e cannot be selected
- e∩f: Means that activities e and f must be selected simultaneously
- e∪f: means that activities e and f must be selected incompatibly

(3) $Restrict_i$ is a set of rules to be followed while assembling the activities in $Event_i$ and these rules restrict the execution sequence of activities. Suppose that the relations between activities are K. Obviously, K is reflexive, anti-symmetry and transitive. Therefore, K is a partial order. Notation < is used to represent a partial order. For example, if $e_1, e_2 \in Event_i$ and $e_1 < e_2$, it means that e_1 must be executed before e_2. These rules are formally described in the following forms:

- Succ(x): A set including all immediate successor nodes of x and formally:

$$Succ(x) = \{ y \mid x<y \wedge \neg(\exists z (x < z \wedge z<y))\}$$

- Prec(x): A set including all immediate predecessor nodes of x and formally:

$$Prec(x) = \{y \mid y<x \wedge \neg(\exists z(y<z \wedge z<x))\}$$

- $Succ^*(x)$: A set including all successor nodes of x. All nodes in $Succ^*(x)$ will be executed after x and formally:

$$Succ^*(x) = \{y \mid x<y\}$$

- $Prec^*(x)$: A set including all predecessor nodes of x. All nodes in $Prec^*(x)$ will be executed before x and formally:

$$Prec^*(x) = \{y \mid y<x\}$$

- First (E): Nodes that can be carried out first among all activities in E and formally:

$$First(E) = \{x \mid x \in E \wedge Prec^*(x) \cap E = \varnothing\}$$

- Last(E): Nodes that can be carried out last among all activities in E and formally:

$$Last(E) = \{x \mid x \in E \wedge Succ^*(x) \cap E = \varnothing\}$$

With the rules above, sequential and parallel structures can be defined. Choice structures are needless, because, when a workflow instance moves to a flexible activity, the activities to be carried out are determinately doubtless. These rules include partial orders internal of loop structures, without regard to the partial orders between nodes that are connected by cyclic control arcs.

S represents the state of a workflow instance and $S \in \{Initial, Running, Wait, Finished\}$. If S = Initial, it means that an initialization work is finished and all activities have not been implemented. If S = Running, it means that a workflow instance is moving and no flexible activity needs to be reified. If S = Wait, it means that a workflow instance moves to flexible activities and sub-workflows need to be reified. If S = Finished, it indicates that a workflow instance is finished.

R comes from historical information of a previous workflow instance and it is preset on the initialization of a workflow instance. The information in R is used to help reifying flexible activities. In facts, a workflow model is an abstract of business processes, moreover, the operating processes of one organization are stable to some extend (or else it is not seen as a workflow). Therefore, it is quite possible that the flexible activities reifying processes are identical between adjacent workflow instances. The historical information of a prior instance is put into R and this measure might reduce the time in flexible activities reifying processes. However, external environments are changing unceasingly, activities in $Event_i$ and the constraint rules in $Rule_i$ and $Restrict_i$ may be changed dynamically according to actual requirements. Therefore, the information in R needs to be checked for its agreement with current requirements. The information in R can been described formally by:

$$R = \{(A; (N_A, F_A)), (B; (N_B, F_B)) \ldots\}$$

where, A and B are flexible activities in a workflow model, N_A is the set of activities selected from $Event_A$ when reifying A, F_A is the flow relations between activities in N_A. N_B and F_B have similar meaning with N_A and F_A.

CORRECTNESS CHECKING ALGORITHM FOR SUB-WORKFLOWS

When a workflow instance moves to a flexible activity, the information in R is used to construct a sub-workflow. An algorithm is put forward to check the validity of sub-workflows in this section. The algorithm works in three steps. Firstly, it checks the compatibility of all activities in R. Then, it examines the dependence relations between activities in R and tests whether the relations agree with the partial orders in $Restrict_i$. Lastly, it checks whether there is a way from First ($Event_i$) to Last ($Event_i$). If R can pass all inspections, it shows that the sub-workflow is correct. Workflow instances can move via the sub-workflow model. Furthermore, the sub-workflows are preserved to be used in future. This algorithm is described in pseudo code as follows:

Algorithm 1: Check validity( )
{

Step 1: Examine the compatibility of selected activities.
{for every n_i in N {
if $n_i \notin Event_i$ return illegal selection;
if n_i is of type -e, return illegal selection;
if n_i, $n_j \in Event_i$ and $n_j \in N$,
if there is a constraint $n_i \cup n_j$, return illegal selection;
if n_i, $n_j \in Event_i$ and $n_j \notin N$
if there is a constraint $n_i \cap n_j$, return illegal selection}
Go to step 2}

Step 2: Check the dependence relations between the activities in $Event_i$.
{for every (f_i, f_j) in F {
if $f_i \in Prec(f_j) \wedge f_j \notin Succ(f_i)$, return illegal dependence;
if $f_i \notin Prec(f_j) \wedge f_j \in Succ(f_i)$, return illegal dependence;
if $f_i \notin Prec(f_j) \wedge f_j \notin Succ(f_i)$,return illegal dependence}
Go to step 3}

Step 3: Examine the construction of sub-process
{if (there is no way from First ($Event_i$) to Last ($Event_i$))
return illegal subprocess graph;
for every $n_i \in N$, except First ($Event_i$) and Last ($Event_i$){
if (cannot find a way from First ($Event_i$) to Last ($Event_i$) through n_i)
return illegal subprocess graph}
return ok}
}

Note that if sub-workflows are illegal, the algorithm returns illegal messages to users. If sub-workflows are correct, it returns ok.

DYNAMIC PROCESS CONTROL ALGORITHM

Workflow instances move according to a defined workflow model and an algorithm is put forward to control dynamic processes in this section. The algorithm works in following steps. Firstly, the running instance is suspended if workflow moves to flexible activities. Then, it examines the information of R. If R is null, it means that it is the first instance relying on certain schema and there is no historical information to be used. Therefore, it calls for a manual assistance to set up sub-workflows. Afterwards, new designed sub-workflows need to be checked by the function Check validity( ). If R is not null, Check validity( ) is called to check the validity of R immediately. If R is valid, the instance starts to run according to R; if not, the information of R needs to be manually adjusted based on the returned messages from Check validity( ). Lastly, R is checked again by Check validity( ) and the process is not repeated until the function returns ok. The algorithm is described in pseudo code as follows:

Algorithm 2: AdaptiveFlow( )
{for all (n_i, n_j) $\in$ F{if (I.S! = Finished $\wedge$ n_i.State == Completed $\wedge$ n_j.Type == Flexible)
/*suspend the workflow instance*/
I.S = Wait;
/*if R is empty, set up sub-workflows manually*/
if (I.R is null) Callman ( )
/*check the validity of sub-workflows*/
else Check validity(R);
/*make R valid and store R*/
do {if (Check validity(R))
Store(R);
I.S = Running;
else Callman( ); }
until Check validity(R) return ok;
}
}

CASE STUDY

A rapid changing business environment makes the requirements of dynamic workflows increase promptly. Dynamic workflow systems are widely used in modern manufacture industries. Here, we illustrate the proposed method by using a developing process of mobile phones. There are more than 200 components in a mobile phone and these parts may be produced by different departments of a company, even by multiple companies in different countries. The developing process of mobile phones is very complex and time consuming and the whole process is packed with uncertain factors.

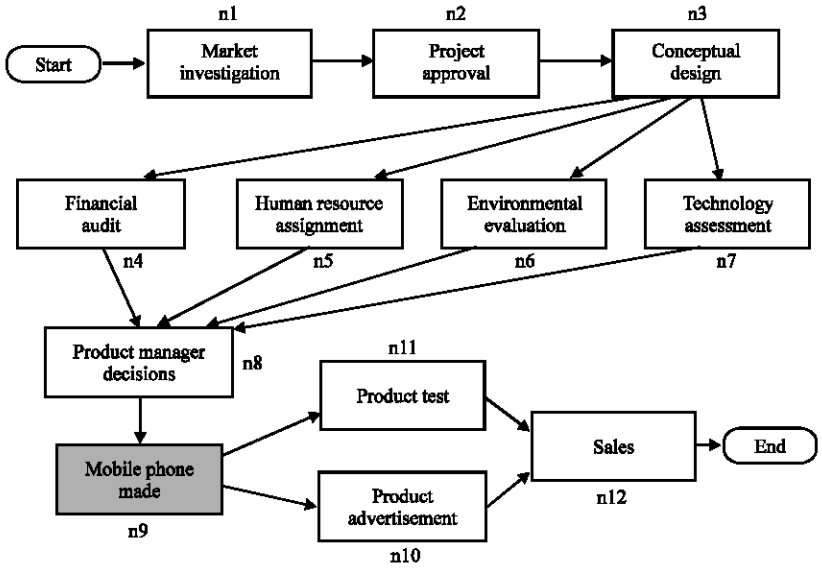

Fig. 1: Procedure of developing mobile phones

Only the approximate frame and major activities are known in the developing process. For example: market investigation, project approval, proposal design, technology assessment, environmental evaluation, etc. These activities are pediocratic in the developing process and complete definitions of them can be given at the stage of initialization, therefore, they are treated as general activities in this example. However, the complete definition of a key activity, Mobile phone made, cannot be given beforehand. Therefore, mobile phone made is seen as a flexible activity. The reason is that different kinds of products can be produced by different departments or different subsidiary companies. The detailed definition of mobile phone made will be clear when the workflow comes to this flexible activity. Figure 1 shows the procedure of developing mobile phones.

The gray activity in Fig. 1 shows the making process of mobile phones and it is a flexible activity. Supposed that the following conditions are satisfied when reifying the activity mobile phone made:

- $Event_9$ includes the following activities: a is A moulds company; b is B moulds company; c is A chips making department; d is B chips making department; e is A batteries making department; f is B batteries making department; g is A embed system company; h is B embed system company and k is products are assembled and installed
- Compatibility between activities. If a is chosen, then c and f must be selected. If b is chosen, then d and e must be selected. a and b must be chosen exclusively. k must be chosen. One of g and h can be chosen randomly

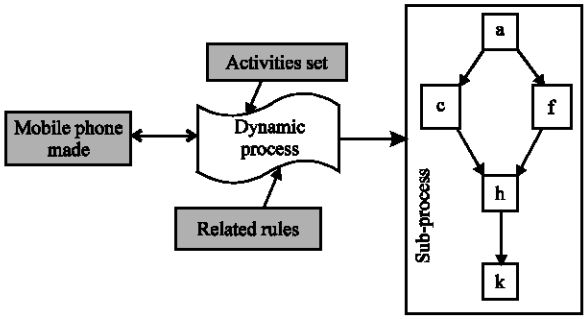

Fig. 2: Reifying process of making a mobile phone

- If a is chosen, c and f must be executed after a. If b is selected, d and e must be executed after b. g or h should be executed just ahead of k and k is the last activity.

Figure 2 shows a reifying process of mobile phone made.

With the dynamic workflow model defined in previous sections, a complete definition of developing mobile phones is shown in as follows.

- ID = NO. 090126
- M = (Pete, 01-26-2009, 2.0, New mobile phone developing)
- Q = (N; F)

N = {n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12};

F = {(n1, n2), (n2, n3), (n3, n4), (n3, n5), (n3, n6), (n3, n7), (n4, n8), (n5, n8), (n6, n8), (n7, n8), (n8, n9), (n9, n11), (n9, n10), (n10, n12), (n11, n12)}
n1 = {Market investigation, General, null, Notactivied}
n2 = {Project approval, General, null, Notactivied}
n3 = {Conceptual design, General, null, Notactivied}
n4 = {Financial audit, General, null, Notactivied}
n5 = {Human resource assignment, General, null, Notactivied }
n6 = {Environmental evaluation, General, null, Notactivied}
n7 = {Technology assessment, General, null, Notactivied}
n8 = {Products manager decisions, General, null, Notactivied}
n9 = {Mobile phone made, Flexible, Cr_9, null}
Cr_9 = ($Event_9$, $Rule_9$, $Restrict_9$)
$Event_9$ = {a, b, c, d, e, f, g, h, k}
$Rule_9$ = {a∪b, a∩c∩f, b∩d∩e, +k, g∪h }
$Restrict_9$ = c ∈ Succ (a), f ∈ Succ (a), g ∈ $Prec^*$(k), h ∈ $Prec^*$(k), First ($Event_9$) = {a, b}, Last ($Event_9$) = k, d ∈ Succ(b), e ∈ Succ(b)}
n10 = {Products advertisement, General, null, Notactivied}
n11 = {Products test, General, null, Notactivied}
n12 = {Products sales, General, null, Notactivied}

- S = Initial
- R = {(n9 ;(N_9, F_9))}

N_9 = {a, c, f, h, k}
F_9 = {(a, c), (a, f), (c, h),(f, h),(h, k)}

This is the initialing information of instance I new mobile phone developing. R comes from the historical information of a earlier mobile phone developing process that depends on the same workflow model.

Workflow instance I starts to move after the initialization works and I.S is set to be running. Activities of instance I are carried out one by one according to the workflow model that I depended on. When n8.state = completed, it starts to reify the flexible activity n9 and AdaptiveFlow ( ) is called to help realizing the dynamic process. Preconditions of reifying n9 are checked at the first step and assume that it is satisfied here. Then, I.S is assigned to be wait and workflow instance NO.090126 is suspended. Secondly, Check validity( ) is called to check the validity of R. R = {(n9; (N_9, F_9))} and it comes from the historical execution information of a previous instance. Function Check validity( ) checks the validity of the information in R and it returns whether R agrees with current constraint rules. If R is not valid, a manual assistant modification process is needed and the revision process is not repeated until R is valid; if R is valid, the instance starts to run by following the information in R. R is supposed to be valid in this example. Moreover, the information of R is stored to be used in the following instances. Lastly, I is resumed and the value of I.S is set to be Running. Instance I continues to move before the end of this developing procedure.

CONCLUSION

In order to improve the capability to respond dynamically to process changes, this paper proposes a dynamic workflow model based on flexible activities and historical information. Each flexible activity is used to encapsulate a group of indeterminate factors, including the constraint rules and optional activities that to be used when reifying flexible activities. Historical execution information is the executive logging of a previous workflow instance. Two algorithms are put forward to guarantee the correctness of sub-workflows and the global control of dynamic process. A developing process of mobile phones is given as an instance of the introduced workflow model. This method can be widely used in modern manufacture industries, especially for the businesses that might vary frequently. However, this model ignores the loop structures in sub-workflows and the flexible activities reifying processes are not intelligent enough. These issues will be investigated in future.

ACKNOWLEDGMENTS

This study was supported in part by the National Natural Science Foundation of China under Grants No. 60773034, 90818023, 90718012 and 60803032; the Scientific and Technological Developing Program of Shandong Province of China under Grant No. 2008GG30001024; the Taishan Scholar Construction Project of Shandong Province, China and the Open Project of the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences under Grant No. SYSKF0804.

REFERENCES

Agostini, A. and G. DeMichelis, 2000. A light workflow management system using simple process models. Int. J. Collaborative Comput., 9: 335-363.

Deng, S.G., Z. Yu and Z.H. Wu, 2004. Research and design of dynamic workflow modeling method. Comput. Integr. Manuf. Syst., 10: 601-608.

Rinderle, S., M. Reichert and P. Dadam, 2004. Correctness criteria for dynamic changes in workflow systems-a survey. Data Knowledge Eng., 50: 9-34.

Sadiq, S., W. Sadiq and M. Orlowska, 2001. Pockets of flexibility in workflow specification. Proceedings of 20th International Conference on Conceptual Modeling, 2001, Yokohama, Japan, Springer, pp: 513-526.

Sun, R.Z. and M.L. Shi, 2003. A process meta-model supporting dynamic change of workflow. J. Software, 14: 62-67.

Van der Aalst, W.M.P. and S. Jablonski, 2000. Dealing with workflow change: Identification of issues and solutions. Int. J. Comput. Syst. Sci. Eng., 15: 267-276.

Van der Aalst, W.M.P. and T. Basten, 2002. Inheritance of workflows: An approach to tackling problems related to change. Theor. Comput, Sci., 270: 125-203.

Yang, S.X. and J. Wang, 2008. Workflow instances migration approach based on state. Comput. Integr. Manuf. Syst., 2: 14-14.

Zhou, J.T., M.L. Shi and X.M. Ye, 2005. State of arts and trends on flexible workflow technology. Comput. Integr. Manuf. Syst., 11: 1501-1510.