Advances in Agile Manufacturing – ICAM 2003

A Flexible Workflow Meta-model Supporting Dynamic Modification

Zhijun Zhang, Yunshun Fan
Department of Automation, Tsinghua University, Beijing 100084, P. R. China

ABSTRACT: Traditional workflow models lack of the necessary flexibility for the complex and dynamic characters of contemporary business processes and thus they limit the application of workflow technology in wider areas. This paper introduces a flexible workflow meta-model which is capable of tackling the problems related to business events based synchronization, multiple activity instances, service type activities and run-time process modeling. This meta-model is approached by extending the traditional activity network model with modeling constructs such as wait connectors, parallel blocks, services and modification blocks. The technique for dynamic modification based on the meta-model is also addressed. Practical examples show that the meta-model has significant advantages like strong expressive power, flexible modeling methods, and clear and concise model structures.

Key Words: workflow model, flexibility, meta-model, dynamic modification

1 INTRODUCTION

Workflow technology, one of the crucial technologies for contemporary enterprises in implementing the management and control of business processes, is increasingly used in a variety of industries, such as manufacturing, business, services and so on [1, 2, 3]. However, as the business processes become more complex and changeful than ever before, the drawback that traditional workflow models are too rigid to describe and enact the actual business processes becomes more and more conspicuous [2, 4]. For example, complex business processes cannot be well described in many workflow management systems and also dynamic changes are managed in unsystematic ways. Such defects limit the application of workflow technology in wider areas. Therefore, flexibility comes to be one of the major research topics in the area of workflow technology [2, 5, 6].

This paper focuses on the workflow meta-model level to improve the flexibility of workflow management systems by supporting flexible modeling and dynamic modification. Many current workflow meta-models do not have enough flexibility especially in the following aspects:

(1) Business events based synchronization

Unlike workflow events (internal events), which are related to the control flow of the process instances, business events (external events) are more often related to the business contexts, such as the arrival event of a cargo posted by the warehouse management system. Many workflow meta-models do not have the complete constructs to describe business events which act as bridges between different process instances and other systems.

(2) Multiple activity instances

In actual business processes, there may simultaneously exist several instances of one activity in a process instance. For example, a number of applications are received, and each of them needs to be handled by the same activity. Traditional workflow meta-models just describe such scenario in a rigid way, that is, coping as many as needed number of activities paralleling to the first one. However, it will not work if the number of instances cannot be determined until run-time.

(3) Service type activities

As modern business processes become customer-take-charge and virtual enterprises become prevalent, services are frequently used in business processes. Service type activities are executed like multiple activity instances, but are always triggered by business events.

(4) Run-time process modeling

Today's workflow management systems typically support a more or less predetermined process model, and do not allow any more dynamic modifications. However, decisions about how some activity definitions in the process model should be are frequently made until the last minutes, and process models defined during build-time also need to be constantly reviewed, improved and adapted.

The flexible workflow meta-model introduced in this paper is capable of tackling the above problems by extending the traditional activity network model

with modeling constructs such as wait connectors, parallel blocks, services and modification blocks. This paper is structured as follows. In Section 2, the workflow meta-model is introduced and its constructs are illustrated in detail. Section 3 presents a sample scenario. The methods of dynamic modification are discussed in Section 4. Finally, Section 5 summarizes our research.

2 WORKFLOW META-MODEL

The meta-model is developed based on the traditional activity network model. Workflow models are represented as graphs where nodes in the graph represent process activities or tasks, and edges depict the flow or the order of the tasks involved in the process. The graphical representations of the modeling constructs are shown in Figure 1.

2.1 Transition

Transitions depict the simple sequential relationship between the two nodes connected by arrow-headed edges. One node has one input transition and one output transition except for symbols, join connectors and split connectors. Transitions can be conditional when they are output transitions of OR-split connectors and XOR-split connectors, and in this instance, they are marked with black ovals on the edges. Unlike traditional workflow models, which define the conditions as properties of the transitions, we define them as properties of the connectors to bring the uniform manipulations when the process model is modified. Data transitions are not defined because they are always invalid without the underlying control flow.

2.2 Activity

An activity represents a piece of work that will be processed by some performers with certain resources to achieve the ultimate goal of a business process.
An activity is atomic and self-contained. The internal structure of an activity is given in Figure 2. The attributes can be business events, priorities, time constrains, narrative descriptions etc., and some of them may be determined during run-time by evaluating the specified expressions.
Activities can be classified into two categories as manual activities and automatic activities. An activity is automatic if it is processed by workflow-enable IT applications, that is, applications which can be handled by workflow enactment systems; otherwise, the activity is manual, and in such instance, it is processes by some organization units perhaps with the help of IT applications. Activities can post external events during execution as specified in the business events attributes, thus enabling the communication between different process instances or business systems.

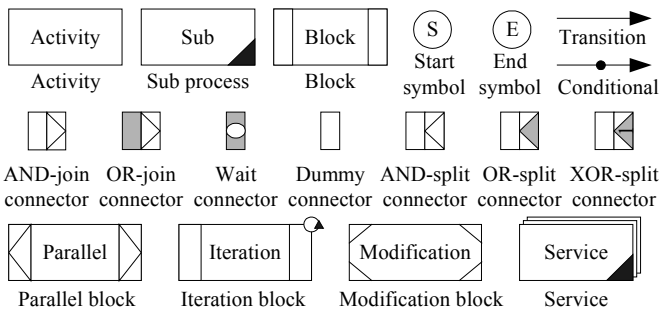

Figure 1. Modeling constructs of the workflow meta-model

2.3 Symbol

2.3.1 Start symbol

A start symbol serves as the only entry point of a process model, a block or a sub-process with no preceding task and one subsequent task (if more than one, preceded by a split connector).

2.3.2 End symbol

An end symbol serves as the only exit point of a process model, a block or a sub-process with one preceding task (if more than one, followed by a join connector) and no subsequent task. When the end symbol of a process instance is reached, which means the instance is completed, activities that are still active in the instance, will be cancelled and their performers will be notified.

2.4 Connector

In many workflow meta-models, for example, the WPDL, relationships and constraints between different tasks such as join, split and synchronization are defined as parts of the activity specifications which cause that any change made to them will affect the activity specifications. In our workflow meta-model, tasks are mediated by discrete constructs named connectors to minimize the effect of such changes through separating the specifications of control information from activity specifications. Connectors together with transitions form the control flow of the process models.

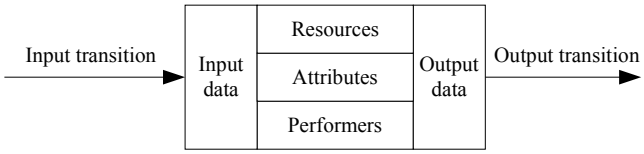

Figure 2. The internal structure of an activity

2.4.1 Join connector

This connector deals with the join relationship in business processes. Join connectors are preceded by several tasks, and followed by one task. Join connectors are classified as:

(1) AND-join connector

It functions as an "AND" logical operation, which requires all of its preceding activities to be finished to activate its subsequent activity.

(2) OR-join connector

The OR-join connector has an OR-join-end condtionwhich is evaluated with return values of "True" or "False" when each of its preceding activities is finished. Only when the condition returns "True", is its subsequent activity activated. If there are still other preceding activities finished after the activating, the subsequent activity will not be activated for another time. The OR-join-end condition can be arbitrary complex. For example, supposing an OR-join connector has three preceding activities, and I[n] denotes the state of the nth preceding activity which returns "1" when the activity is finished, and otherwise "0", expression "I[1] AND I[3]" means the subsequent activity is activated when the first activity and the third activity are finished; expression "I[1]+I[2]+I[3]>=2" means the subsequent activity is activated when there are no less than two activities are finished.

2.4.2 Split connector

This connector deals with the split relationship in business processes. Split connectors are preceded by one task, and followed by several tasks. Split connectors are classified as:

(1) AND-split connector

The AND-split connector activates all its subsequent activities when the preceding activity is finished.

(2) OR-split connector

This connector enables the function to selectively activate subsequent activities. The OR-split connector has a list of conditions that return values of "True" and "False" corresponding to its conditional output transitions. After the preceding activity ends, the conditions are instantaneously evaluated and only subsequent activities with "True" condition values are activated.

(3) XOR-split connector

The XOR-split connectors, a special sort of OR-split connectors, permit only one of the subsequent activities to be activated. Since it is always too complex to judge whether the conditions are exclusive, only the first activity is activated when several conditions return "True". Further more, exception handling methods can be used to tackle this problem.

2.4.3 Wait connector

The wait connector is a time-consuming modeling construct that performs the function to wait for the satisfaction of a condition, the arrival of a specific time or a time-span and the occurrence of an event which are together denoted by a wait-end condition. The condition is evaluated all the while until it becomes "True", and then the subsequent activity is activated. The wait connector can be used to synchronize different activities in the process instances. Sometimes, the wait-end condition may never be satisfied, so including time constraints in it is always considered as a good design.

2.4.4 Dummy connector

Dummy connectors are routing marks without any actual function. They are always used to meet the requirements of clarity or aesthetics in graphic design tools.

2.5 Sub process

Sub processes are a type of modeling constructs which provide the mechanism for building hierarchical process models and support the top-down modeling methods. A sub process specification shares a lot with a process model. Both are composed by a set of activities and connectors; both have some transactional semantics and exceptional behaviors. However, sub processes are always considered as unfledged process models because they cannot be directly instantiated, which means they must be embedded in certain process models. Sub processes are clearly specified with a public interface (input and output parameters) and a public objective (its description), which facilitate the reuse of them in many process models as building blocks.

2.6 Blocks

2.6.1 Block

The concept of blocks is so similar to that of sub processes that they can be considered as a type of sub processes which are not reusable. Unlike sub processes which use public interfaces to specify the data flow, blocks use input and output data attributes the same as common activities.

2.6.2 Iteration block

Iteration blocks are introduced to meet the requirements of multiple activity instances executed sequentially. In addition to the normal block specification, an iteration block definition includes a repeat-until condition, which determines at runtime how many times the internal block is executed continuously. The execution scenario of an iteration block is given in Figure 3.

The input and output data of an iteration block which are used by all the internal block instances are created as global data of the iteration block upon entering the first execution. As an example, consider a document review process which accepts a raw

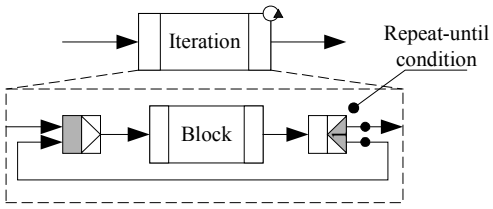

Figure 3. Iteration block

document as input data and completes with a fine one through continuous reviews and modifications. During run-time, the iteration block creates a document as global data, then the document is available for all the internal block instances to read and modify until the repeat-until condition is satisfied when the review step results with "OK".

2.6.3 Parallel block

Parallel blocks are always used where the tasks executed in parallel are similar to each other, and the number of tasks cannot be determined until run-time. A parallel block definition includes a parallel-end condition, which determines when the parallel block instance can be considered as completed, since some of the internal block instances may be still being executed. The execution scenario of a parallel block is given in Figure 4.

Obviously, the input and output data of a parallel block which are used by the internal block instances should be created as data list type. During run-time, the number of the internal block instances to be instantiated is determined by the dimension of the input data list, and the dimension of the output data list equals to the number of the normally completed internal block instances which depends mainly on the parallel-end condition that is evaluated when each of the internal block instance is finished. For example, supposing a parallel block pb, DI[pb] denotes the dimension of the input data list of pb, DO[pb] denotes the current dimension of the output data list of pb, and pb[i] denotes the ith internal block instance

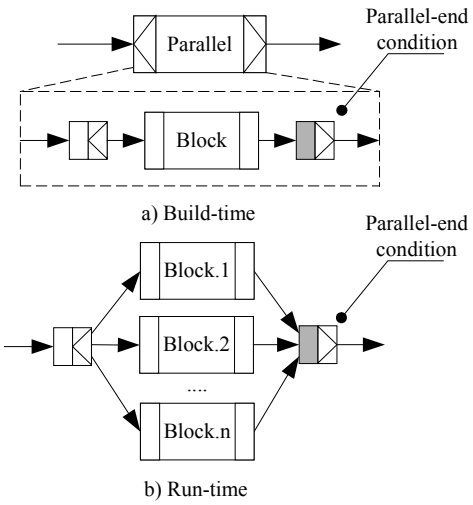

Figure 4. Parallel block

with 0< i <=DI[pb], I[pb[i]] denotes the state of pb[i] which returns "1" when it is finished, and otherwise "0", expression "(DO[pb]>=1) AND (DI[pb]-DO[pb]<=2) AND (I[pb[1]] = 1)" means the parallel block is completed when there are no less than one finished internal block instance and no more than two unfinished internal block instances, and the first internal block instance must be finished. The parallel-end condition can be quite meaningful by involving more actual attributes of the parallel block.

2.6.4 Modification block

Since business environments are continuously changing in a rapid pace, it is almost impossible to identify all the control steps before run-time. Decisions about how some activity specifications and their relationships in the process model should be are frequently made until the last minutes. Modification blocks are such a type of modeling constructs that enable users to sketchily design process models at build-time, and detail them in the last minutes with automatic notifications as common work items. When the modeler submits the modifications, the interrupted process instance resumes from the start symbol of the internal block of the modification block. Usually, the modifications are limited to the current process instance. But the modeler can apply them to the process model if necessary. The modification methods are introduced in the following section.

Modification blocks are especially competent for two circumstances. The first one is "trial-and-error" like process where the specifications of predefined control paths make no sense, and the relationship of tasks must be configured during run-time. The second one is "self-learning" process, which construct the process models from blank by gathering information based on repetitious execution.

2.7 Service

With modern business processes become customer-take-charge and virtual enterprises become prevalent, a great number of workflow models contain service type activities. As given in Figure 5, a service definition includes an event listener, which creates an instance of the internal sub processes on receiving the designated event(s), and a service-expire condition, which determines at runtime when the service instance should be ended. Like parallel blocks, the internal sub processes of services can be instantiated for many times, however, there are several differences:

(1) Because the processes of services are always mature and reusable, so they are developed base on sub processes, not blocks.
(2) The instances are always created corresponding to the coming event(s) of services, and the number of instances is uncertain even during runtime.

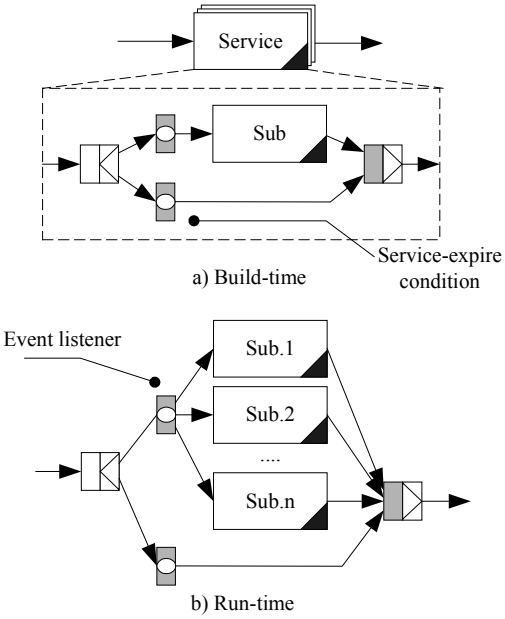

Figure 5. Service

(3) In parallel blocks, the input data are internal data of the current process instance, however, in services, the input data are often external data and are delivered together with the event(s).
(4) Unlike the parallel-end condition, the service-expire condition always contains temporal information, and is evaluated all the time until it is satisfied.

A "call for papers" process is a good example for services. After a "call for paper" bulletin is announced, the paper collecting service begins. Every paper arrived posts a "paper-arrival" event to the service instance, then the service instance creates a separated process to receive the paper, for example, to preview and preserve it. When the deadline arrives, the service expires. All the papers collected are ready to be processed in further steps.

3 A SAMPLE SCENARIO

To illustrate the approach, an introductory example based on a bidding process is given in this section. All the process models involved are depicted in Figure 6.

The bidding process starts with preparing documents step, which is described as an iteration block that involves amending (drafting) the tender documents and examining them. This task will not be finished until the examining step returns "OK". Then, the "call for bids" task releases the information to the potential participants and a receiving bids service with a deadline type service-expire condition begins after that. Once a new bid arrives, an internal sub process instance of the service will be triggered by the very event. After being checked, the bid is registered in the system if it is qualified. Otherwise, the bidder is informed with the detail reasons. The service will be valid till the deadline arrives. After that, if there is no bid (or not enough bids) for further evaluation, a warning is sent out, and the process ends. In the other way, preliminary evaluation consisting of the analysis step and the evaluation step will be executed for every bid as a parallel block whose parallel-end condition requires all the instances to be normally finished. With some bids are rejected as the results of this step, others will be processed by the following tasks which includes final evaluation, awarding of bid and announcing the winner. Note that the final evaluation task is described as a modification block which will be detailed during run-time since it is difficult to determine the exact structure during build-time. Assuming the final evaluation step is originally represented as an empty block, then, during run-time, the modeler constructs it by adding two parallel parts: financial analysis and evaluation, technical analysis and evaluation. The modeler can elaborate the technical evaluation task further more if he finds that the technical evaluation task can be done either by an internal team, by a consultant group, or both. Finally, the bidding process will be concluded with the contracting task.

From the example above, the advantages of the proposed workflow meta-model can be evidently recognized:

(1) Strong expressive power

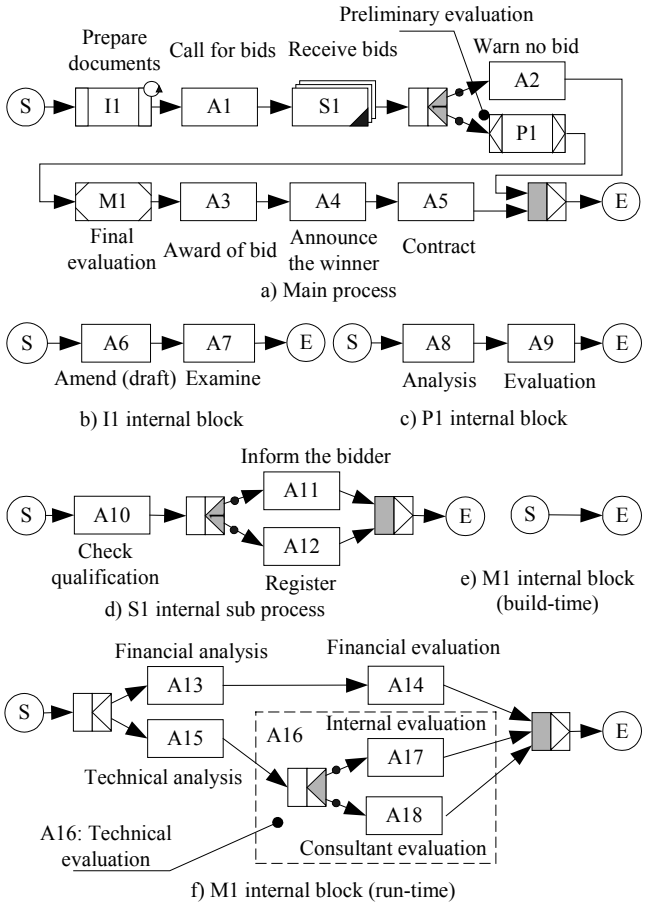

Figure 6. Bidding process models

Complex business processes with business events, multiple activity instances and services which are not supported or ill defined in traditional meta-model are well represented with the given modeling constructs. Also, the workflow meta-model allows the condition attributes of the constructs to be arbitrarily complicated.

(2) Flexible modeling methods

Model structures can be depicted in more flexible ways and even be detailed in the last minutes which facilities the usage of workflow management systems in ad-hoc processes and self-learning processes which have ever been notorious in the application of workflow technology.

(3) Clear and concise model structures

Control flows are clearly represented with elements such as symbols, connectors, and transitions, and the model structures are abbreviated with special blocks and sub processes. Yet the separation of connectors and transitions may incur complexity of the model structure, it is necessary for clear descriptions of the diversiform relationships between different tasks.

Data transitions are not defined because they are always invalid without the underlying control flow.

4 DYNAMIC MODIFICATION

Dynamic modifications are always accomplished through the APIs provided by workflow enactment systems which allow the modifications of the runtime control structures and also ensure the correctness of the process models. Actually, modifications to an existing model by separate operations which ensure the correctness for each alone are discouraged by the large amount of the potential operations and also the intricate interactions between them. A feasible way is to define some meta-APIs for the modifications and keep correctness in a high level, namely the operations set transaction level. The modeler use the APIs to modify the process model and then the workflow enactment systems check the result based on some correctness rules before it is eventually submitted.

4.1 Meta-APIs

The meta-APIs are designed in term of the desirable properties as: completeness and correctness. The meta-APIs consist of operations to create and delete all of the workflow constructs including transitions, activities, symbols, connectors, sub processes, blocks and services. Replacing operation can be achieved by the combination of both deleting and creation operations. Also, the meta-APIs contain operations to modify the attributes of the constructs as well. The following is the COM IDL format definition of the meta-APIs. In this definition, the abstract entity concept is introduced to represent all types of the node constructs. In addition, complex data structures such as entity and attributes list are described in the XML format.

```
interface Iwf Process Modification:
{
   HRESULT Create Entity ([in]long Proc ID,
[in]short Entity Type, [out]long* Entity ID);
   HRESULT Delete Entity ([in]long Proc ID,
[in]long Entity ID);
   HRESULT Get Entity ([in]long Proc ID, [in]long
Entity ID, [out]BSTR* xml Entity);
   HRESULT Get Entities List ([in]long Proc ID,
[in]BSTR Filter, [out]BSTR* xml Entities List);
   HRESULT Get Entity Attributes List ([in]long
Proc ID, [in]long Entity ID, [out]BSTR* xml Attribs
List);
   HRESULT Assign Entity Attributes List ([in]long
Proc ID, [in]long Entity ID, [in]BSTR xml Attribs
Lsit);
   HRESULT Add Transition ([in]long Proc ID,
[in]long From Entity ID, [in]long To Entity ID)
   HRESULT Remove Transition ([in]long Proc ID,
[in]long From Entity ID, [in]long To Entity ID)
      …
}
```

4.2 Correctness rules

In order to prevent incomplete models or incorrect instances resulted from random modification operations, constraints of each set of operations should be assured. The rules here mainly concern to the behavioral perspective of the meta-model:

(1) The meta-model does not support the implicit definition of iteration, so the workflow model should be a directed acyclic graph (DAG).

(2) There can only be one transition between two different nodes in a process model.

(3) The number constrains of input and output transitions for each type of nodes must be ensured. For example, one node has one input transition and one output transition except for symbols, join connectors and split connectors.

(4) There can only be one start symbol and one end symbol in a process model, a sub process, or a block.

(5) Any reference to other constructs or their attributes in the attributes of the current construct should be valid. For example, the condition attributes for some type of nodes are always invalidated by the deleting operations of the preceding nodes or the subsequent nodes.

(6) If the input data and output data of activities or blocks, as well as the input parameters and output parameters of sub processes are changed, the

corresponding data need to be modified too. In particular, if the interface of a sub processes is changed, the effective scope may much larger than the current process for its reusability.

(7) Events used in wait connectors and services should be available during run-time.

Since the rules above just guarantee the correctness of the workflow graph, other correctness related to the combinative usage of join and split connectors and the actual condition expression logic should be further involved. For more detailed information about this topic see [7]. In addition, dynamic modifications can be used in any stages during run-time besides the instantiation of predefined modification blocks. But in this instance, another verification process should be introduced to check if they could be applied to the existing instances according to their execution stages [5, 6].

5 CONCLUSIONS

Supporting flexible modeling and dynamic modification becomes absolutely necessary to increase the applicability of workflow management system. This paper introduces a flexible workflow meta-model which is capable of tackling the problems relating to business events base synchronization, multiple activity instances, service type activities and run-time process modeling, and also has good performances in the modeling structures and modeling methods. It is very feasible to use this workflow meta-model to describe the contemporary business processes characterized by increasingly complex and rapidly changing. Further work should be addressed on other aspects of the meta-model including transaction behavior, exception handling and instances migration.

REFERENCES

[1] Markus Kradolfer. 2000. A Workflow Metamodel Supporting Dynamic, Reuse-Based Model Evolution. Ph.D. Dissertation: University of Zurich.

[2] Fan Yushun. 2001. Workflow Modeling Method Based on Coordination Theory. Proceedings of the IASTED International Conference-Intelligent Systems and Control, 2001: 238-241.

[3] WFMC. 2002. Workflow Process Definition Interface -- XML Process Definition Language (WfMC-TC-1025). Workflow Management Coalition.

[4] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, et al. 2002. Workflow Patterns.

[5] R Siebert. 1999. An Open Architecture for Adaptive Workflow Management Systems. Society for Design and Process Science.

[6] Shazia W Sadiq, Maria E O Rlowska. 1999. Architectural Considerations in Systems Supporting Dynamic Workflow Modification. Proceedings of the Workshop on Software Architectures for Business Process Management at CaiSE99.

[7] Wasim Sadiq, Maria E. Orlowska. 2000. Analyzing Process Models Using Graph Reduction Techniques. Information Systems, Vol.25, No.2, 2000: 117-134.

[8] Gabriel Sanchez Gutierrez. 1997. The WIDE Workflow Model and Language. ESPRIT Project: Workflow on Intelligent Distributed database Environment.

[9] Jie Meng, Stanley Y W Su, Herman Lam, et al. 2002. Achieving Dynamic Inter-organizational Workflow Management by Integrating Business Processes, Events, and Rules. Proceedings of the Thirty-Fifth Hawaii International Conference on System Sciences.