

# **A Framework for Resource-based Workflow Management**

**Akhil Kumar**

Smeal College of Business,  
Pennsylvania State University, University Park, PA 16802, U.S.A.,  
[AkhilKumar@psu.edu](mailto:AkhilKumar@psu.edu).

**Jianrui Wang**

Department of Industrial and Manufacturing Engineering,  
Pennsylvania State University, University Park, PA 16802, U.S.A.,  
[JerryWang@psu.edu](mailto:JerryWang@psu.edu).

## **Abstract**

This chapter presents a general framework of resource-driven workflows as an alternative to the more popular control flow driven workflows approach. We argue that this approach is more holistic than control flow driven approaches because it considers availability of resources such as data, people, equipment, space, etc. Control flow driven approaches usually either disregard resource considerations or account for them only implicitly. In our approach the control flow is a derivative of the resource needs of various tasks. Moreover we make a clean separation between hard constraints that arise from resource considerations and soft constraints that result from business policy. The new methodology for process design is described at length, along with an architecture and a detailed discussion of implementation issues. This approach is more holistic and is particularly suited for ad hoc workflows as opposed to production workflows.

## **1 Introduction**

There are many approaches and frameworks for designing business workflows (see [1, 11, 12, 24, 26, 27, 31]). Most of them are based on mapping a control flow that specifies the coordination of various activities. Here we discuss another

approach, which differs from the control flow driven workflow approach, called *resource-driven workflows*. The main idea is to design a process so that the tasks within it can be driven based on the availability of resources required to perform them without an explicit control flow. In general, a process contains several tasks, and each task requires resources like data, people performing roles, equipment, facilities, etc. for its completion. If any resource is not available then the task cannot be performed. We argue that in some scenarios, particularly those involving frequent changes in the environment, as well as resource intensive or ad hoc workflows, the control flow driven approach to workflow design is less suitable. Instead since exceptions and changes are more likely to occur in these cases, a resource-based approach may be more promising. From a management standpoint as well, in recent years the resource-based view of organizations is assuming greater importance as a basis for developing competitive business strategy [8].

In a conventional control flow based workflow, the coordination among various tasks is prespecified using constructs like *sequence*, *choice*, *parallel* and *loop*. More advanced constructs or patterns [3] may also be used. Thus, a sequence construct between tasks A (e.g. 'receive order') and B (e.g. 'check order') creates a dependency between them which means that B cannot start unless A is finished. Now, in general, such a dependency could be for a variety of reasons. It could be because B needs the data produced by A. It could also be because A and B are to be done by the same individual. It could be because A and B need the same facility or equipment. Finally, it could be because of a business policy in the organization. Thus, a process design based on a control flow creates dependencies between tasks, but it does not give a reason for them.

Similarly, consider another scenario where two tasks C (e.g. 'check customer credit') and D (e.g. 'check inventory') are in parallel in the control flow of a process. This means that they do not have a dependency between them. During execution of the process, it may turn out that, in general, both C and D might well need the same human resource, say, a manager, and since there is only one person available in that role, both C and D cannot be done in parallel. They might also need some equipment of which there is only one instance. Hence, this suggests that often because of resource conflicts it is not possible to design a control flow without knowledge of the resource requirements of the various tasks and the available resources. Since the available resources change dynamically there is some value in not "hard-coding" them into the process design. Thus, in this situation, we cannot really say whether C and D are in sequence or in parallel. Consequently, a resource-based approach to process design and execution may be more useful.

As an example to motivate the need for resource-based workflow modeling, consider the new product development process in a company. Such a process involves steps such as product planning, conceptual design, component design, overall assembly, prototype, performance test, etc. In each step individuals from different departments (e.g., marketing, engineering, and development) performing various roles (such as designer, engineer, manager, etc.) are involved (see Table 1). The main features of this process are that the tasks have complex coordination

and routing requirements, and must be routed among individuals or teams which may be geographically distributed. They may need to share documents and other resources. Finally, access to all documents must be carefully controlled based on permissions

Table 1: An example of tasks and their resource needs in product design

<b>Task</b>	<b>Data Resource</b>	<b>Human Resource</b>	<b>Physical Resource</b>	<b>Equipment Resource</b>
Product planning	<u>In:</u> marketing report <u>Out:</u> design spec.	Marketing, design manager	Conference room (capacity 10)	White board PC projector
Conceptual design	<u>In:</u> design spec. <u>Out:</u> detailed design	Design engineer	Design room	Design PC
Design Review	<u>In:</u> detailed design <u>Out:</u> discussion transcript	Design team	Conference room (capacity 25)	White board PC, projector
Component Design	<u>In:</u> detailed design, transcript <u>Out:</u> drawings.	Manufacturing engineer	Office room	CAD workstation

In this table there are four types of resources. The successful completion of a process requires coordination among all the resources. In a resource-driven workflow the various tasks can be scheduled only when all their resource needs are satisfied. Thus it is not necessary to specify a control flow before hand, but it is necessary to specify all the tasks that must be performed and the resource requirements for each task. Document- and entity-centric approaches for modeling business processes are discussed in [5, 6, 10, 14 17, 20, 30].

The objective of this chapter is to present an alternative away of designing workflows. The proposed resource-driven workflow framework is useful when multiple resources are involved in a process and resource conflicts are likely to arise. Therefore, it becomes necessary to look beyond a control-flow centric approach. In addition, this framework can also be used to generate a preliminary design for ad hoc workflows, which may be refined further to create a final process design. The organization of this chapter is as follows. We will first provide some background and contrast resource-driven workflows with control-driven ones in Section 2. Next in Section 3, we will discuss the resource-driven approach in detail. Later, Section 4 gives a general framework for developing resource-driven workflows and an algorithm for handling exceptions, while in Section 5 a comparison between the two approaches is conducted. Section 6 provides a detailed discussion and Section 7 concludes the paper.

## 2 Background

### 2.1 Resource dependencies

Conventional workflow systems emphasize the control flow of a process, that is, the execution sequence of the various tasks. Control flow diagrams assume that the process designer possesses the business domain knowledge to layout the task sequence without addressing the resource requirements of each task. Task sequencing follows from the various kinds of dependencies that exist between them. Malone [19] summarizes three basic types of dependencies that arise in collaboration enterprises: *Flow*, *Fit*, and *Sharing*, as shown in Figure 1. A *flow dependency* arises when one activity produces a resource that is used by another activity. *Fit dependencies* occur when multiple activities collectively produce a single resource. In such situations these activities must be synchronized. For example a series of activities are required to process a customer order such as the one shown in Figure 2 (to be described shortly). These various activities must be synchronized. A *sharing dependency* arises when several tasks compete for the same resource, e.g., when two activities need to be done by the same person. It should be noted that current workflow systems are particularly weak in handling sharing dependencies.

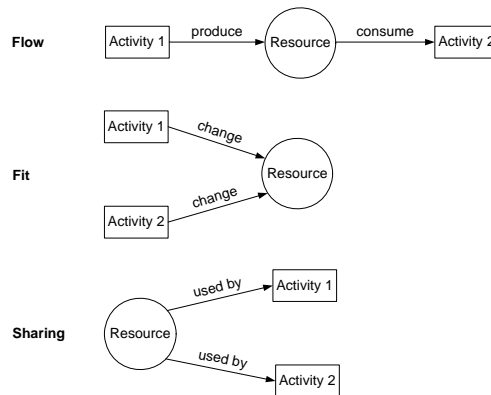
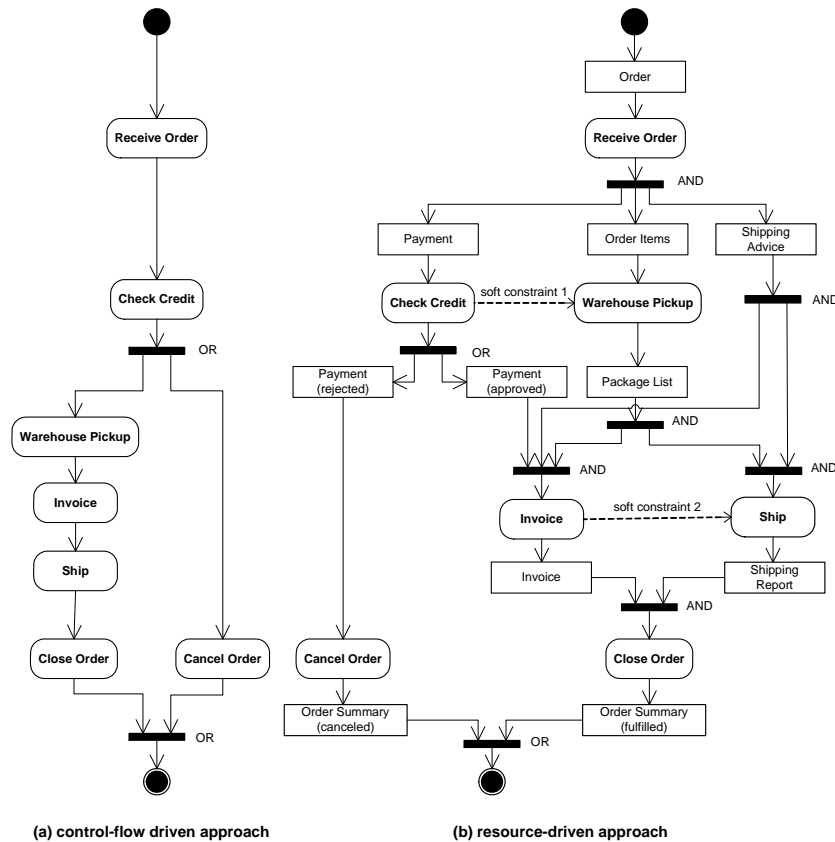


Figure 1: Three basic types of dependencies in any collaboration environment



(a) control-flow driven approach

(b) resource-driven approach

Figure 2: An example order process modeled by control flow and resource-driven workflow

Figure 2(a) shows the main steps in a simple workflow process for handling orders from customers. After an order is received, a *credit check* is performed to verify the payment information. Then there is a *split fork* corresponding to a condition test: if the credit check passes, the order is *picked, shipped, invoiced and closed*; otherwise, it is cancelled. At an AND fork both branches can be taken in parallel, while at an OR fork only one branch can be chosen. Each fork has a matching join node where the branches meet. Each process also has a distinguished start and end node. The shortcomings of this method are that there is no information about resources required for each task such as:

- **Data Resource:** What input documents are needed for the task?
- **Human Resource:** Who will perform the task (a generic role, a team or individual person)?
- **Physical Resources:** physical space/ facilities, etc.
- **Equipment Resource:** PC, video projector, workstation, etc.

In the absence of this additional information, the workflow description is incomplete. Perhaps, the additional information exists in different systems and if so, it will be hard to integrate with the workflow system. Moreover, any attempt at such integration will slow the performance of the system because of the need to exchange different formats and perform transformations. Ideally, a complete or more holistic description must capture such missing information in a common framework. Of course, additional modeling effort is required in the latter case.

Figure 2(b) shows an example to contrast the traditional approach for describing workflows and the resource-driven one. In this example, when we use the term resource the focus is on the document resource. Both parts of the figure show a workflow for order processing. The approach in Figure 2(a) is called the *control-flow-driven approach* because the exact flow of control is specified precisely. In Figure 2(b) the various tasks are shown only with respect to the documents they need, and not as a control flow. Thus, the *receive order* task can only be performed when an order document is available as an input. Moreover, this task produces three output documents: *payment*, *order items* and *shipping advice*. While this figure also looks like a control flow, yet, there is a subtle difference in that the ability to perform a task depends on the availability of the input documents required to perform it. Since a document is a resource, we call this a *resource-driven approach*. Also, a task may require other resources such as people, physical space, equipment, etc. It should be noted that one significant difference between Figure 2(a) and Figure 2(b) is that the former has only an implicit assumption on resource dependencies but the latter makes it explicit.

The resource-driven model can be developed by first conducting an information or data flow analysis as shown in Table 2. This analysis naturally leads to a derivation of the *data dependency constraints* [28]. Such constraints are called *hard constraints* because they are dictated by the resource needs of various tasks. On the other hand, a second type of constraint is dictated by the business policy of the organization, such as the one shown by dotted lines between *check credit* and *warehouse pickup*, and also between *invoice* and *ship* tasks in Figure 2(b). Such constraints are called *soft constraints*.

Table 2: Information flow analysis for tasks in an order process

<b>Task</b>	<b>Input Data</b>	<b>Output Data</b>
<u>Receive order</u>	Order information Payment information (i.e name, customer ID, credit card) Order items (SKUs, unit price, quantity) Shipping information (i.e. Fedex)	The order information in the input document is split into three output documents: Payment, Order items, and Shipping advice
<u>Check credit</u>	Payment	Approved or rejected
<u>Warehouse pickup</u>	Order items	Package list
<u>Invoice</u>	Payment, Package list, and Shipping advice	Invoice
<u>Ship</u>	Package list; Shipping advice	Proof of shipment

Consequently, it is important to make a distinction between these two types of constraints: *hard* and *soft* (see Table 3). A *hard* constraint between tasks A and B arises when task A produces output that serves as input for task B. Hence, B must wait for A to finish (assuming each task is atomic). This gives rise to a strict data dependency between two tasks. Thus, *credit check* can only be done after an order is received. However, *soft* constraints reflect rules in the form of business policy, as opposed to a strict data dependency. So the control flow of a process (see Figure 2(a)) may have a *check\_credit* step to be followed by *warehouse pickup* and *invoice* (if *check\_credit* succeeds). However, the *warehouse pickup* does not require any input data from the output of the *check\_credit* step. Such a business rule can be expressed by a *guard constraint* of the form: *check\_credit.status == "done"*, for the *warehouse pickup* step. This constraint states that the credit must be approved before *warehouse pickup* can start (even though *warehouse pickup* does not require any *specific* input data from the *credit check*). It is shown in Figure 2(b) as *soft constraint 1*. Similarly, the control flow in Figure 2(a) shows that the *invoice* step is followed by the *ship* step. Yet, this again is just a business rule because normally the *ship* step does not need any input from the *invoice* step. This is also represented by a guard constraint of the form: *Invoice.status == "done"*, for the *ship* step. Perhaps the rationale for this constraint might be a company policy that goods cannot leave the company unless the invoice is prepared. It is shown in Figure 2(b) as *soft constraint 2*.

Table 3: Scenarios to illustrate hard and soft constraints between two tasks, A and B

Constraint Type	Description	Example (see Figure 2(b))
<u>Hard</u>	Output of Task A is input for Task B	The <i>check_credit</i> step can only be done after the order is received
<u>Soft constraint 1</u>	There is a guard condition for Task B	<i>check_credit.status == "done"</i>
<u>Soft constraint 2</u>	There is a guard condition for Task B	<i>Invoice.status == "done"</i>

As discussed above, documents that contain data are a resource. Similarly, there are other resources also. One can store resources and availabilities in a database. For doing so, the following data/characteristics should be stored for each type of resource.

- **Document** (*doc\_id*, *description*, *availability*)
- **Human** (*role\_id*, *person\_id*, *time\_period\_id*, *availability*)
- **Space** (*type*, *location\_id*, *description*, *capacity*, *time\_period\_id*, *availability*)
- **Equipment** (*type*, *equip\_id*, *description*, *location\_id*, *time\_slot\_id*, *availability*)

The document resource describes a *doc\_id*, *description* and *availability* (yes/no) at the current time. For a human resource, each tuple contains a role, an id of a person that fills the role, and time periods and *availability* during those time periods. In the case of a space resource we store the type of resource (conference room, office, lecture room, etc.) and its unique location id, along with attributes

such as capacity, and availability during various time periods. For an equipment resource, the schema contains an equipment type and unique id along with attributes like description, location, and availability information. A workflow process consists of tasks. Each task is associated with the resources as follows:

Task(task\_id, task name, doc\_id, role\_id/person\_id, location\_id, equip\_id)

At runtime, a process is instantiated and tasks that are ready to run can be started. The database can be queried to determine if the resources required for a task are available. Thus, if the Warehouse Pickup task needs a Warehouse clerk, then a query on the **Human** resource table can determine if an individual in this role is available before this task can be performed. Similarly, the **Space** and **Equipment** tables can also be queried. As availability of resources changes, the data in these tables is updated dynamically. Incidentally, the assignment of human resources to tasks can be done either in *pull mode* where tasks are offered to individuals and they choose tasks they would like to perform, or *push mode* where tasks are automatically assigned to persons who are qualified for them.

In the next subsection, we will give a resource taxonomy and discuss the difference between instance level and process level resources.

## 2.2 Resource taxonomy

It should be noted that the resources shown in Table 1 have different features. For example, data resources, such as marketing report and design specification, are tightly related to the process instance (or case), and are meaningful only in the context of a specific case because information in these data resources varies from case to case. On the contrary, human resources, such as engineer and manager, may be shared by many cases of a process and can also exist independently of any cases.

In general, resources used in workflow systems can be classified into two classes: a) instance-level and b) process-level (see Figure 3). A document is an *instance-level resource* because it is specific to a case. On the other hand, a human role or equipment is a *process-level (or organizational) resource* since it belongs to the organization and may apply to a variety of instances. A document or data resource triggers a *workitem*, which is an instance of a task that pertains to a specific case. A document is also updated by the case. Thus, a document resource is always "owned" by a case. A process-level resource is not owned by any case. Instead, it enables a workitem to become an activity that is ready for execution. The reason for making this distinction is that these two types of resources are very different. Process level or organizational resources are physical in nature and have implications for scheduling (a human can only do one task at a time), utilization, substitution, etc. There are also business policy considerations like separation of duties, i.e. the same person may not be allowed to perform two tasks in the same process (such as submit a purchase order and approve the purchase). On the other

hand, an instance level resource like a document is non-physical, and there are implications in terms of its ownership, privacy, sharing rules, etc.

This distinction between the two main classes of resources is reflected in the prototype design for a resource driven workflow system discussed in Section 4.

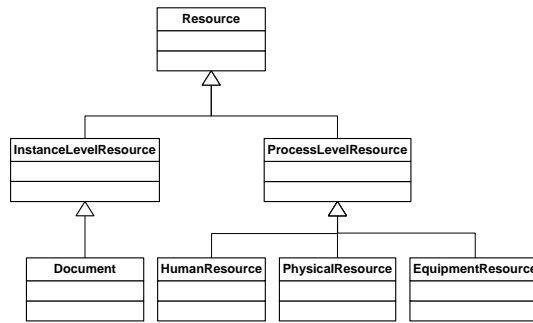


Figure 3: A resource taxonomy

### 3 Proposed approach for designing resource-driven workflows

#### 3.1 Task analysis

As discussed above, the proposed resource-driven approach differs from the control-flow-based approach in that it relies upon understanding the prerequisite resources for each task. The underlying premise here is that by focusing on the requirements for each task, the process flow will emerge organically, rather than being predetermined. The resource driven approach can be modeled as shown in Table 4. This table can be normalized for storing in a database. Here, along with each task, we show the input document it requires, the output document produced by it, the human resource and role that performs the task, the input constraints that must be satisfied (*guard-in*), and the output conditions produced by the task (*guard-out*). Thus, row 2 shows that the *check credit* task is performed by the system automatically. The input for it is the payment information document and the output is the approval number (if credit is approved). The *guard-out* condition is: "*credit == pass*" or "*credit == fail*". The *guard-in* condition is essentially a soft constraint discussed in Section 2, while the *guard-out* condition acts as an integrity check on the output of a task. This example shows us that, in this way it is possible to associate optional *entry* and *exit* constraints for each task based on resource dependencies. Additional columns can be added to the schema of Table 4

to capture needs for other resources. Once a **Schema** table like Table 4 is constructed, then a standard database engine can drive the process flow by running simple SQL queries that find the next task that is ready to run.

Table 4: Schema Table to describe the order processing workflow

Task	Data Resource		Human Resource	Conditions (optional)	
	In_doc	Out_doc	Role	Guard-in	Guard-out
<u>Receive</u>	Order		Order clerk		
<u>Order</u>					
<u>Check Credit</u>	Payment	Approval number	System task		credit == 'fail' credit== 'pass'
<u>Warehouse Pickup</u>	Order items	Package list	Warehouse clerk		
<u>Ship</u>	- Package list - Shipping advice	Shipping confirmation	Shipping assistant		
<u>Invoice</u>	- Package list - Shipping confirmation - Payment	Invoice	Accounts officer	Ship.status == 'done'; Credit == 'pass'	
<u>Close</u>	- Invoice	Close confirmation	Accounts	Invoice.status == 'done'	
<u>Cancel</u>	- Invoice	Cancel confirmation	Accounts	Credit == 'fail'	
<u>Order</u>					

The query will basically return the task(s) for which resources are available. If multiple tasks are enabled, they may be executed simultaneously (or in parallel). Thus, the *parallel* control flow construct of a workflow system is automatically simulated. On the other hand, parallel execution of two enabled tasks can also be prevented by adding a soft constraint as in the example of Figure 2(b). Thus, the entry constraints are applied to these tasks (through guard-in) to determine which task(s) can be executed. Other related SQL queries can be written to find:

- Whether an individual is available in the role required to perform a task?
- What tasks can a role perform?

In general, additional tables would be added to this schema to define mappings between users and their roles (e.g. Jill is a vice-president, Joe is a manager), between teams and their members (e.g., a design review committee consists of a manager and three engineers), etc. The schemas for some of these tables are:

- **Role** (id, role, user\_name)
- **Team** (id, name, member\_role)

Hence, it is possible to assign a task to a team [2] as well, although Table 4 only shows individual roles.

### 3.2 Data dependencies

This section discusses at length the dependency analysis of one important resource type, namely information or data. As noted above, if the input of task B is contained in the output of task A, then task B cannot start before task A finishes. This is the most important kind of data dependency. However, there are other data dependencies as well that are more subtle and should be analyzed. Consequently, in this section we discuss data dependencies in more detail since they play a crucial role in our framework.

We have identified 9 types of data relationships between two tasks, say Task A and Task B, as shown in Table 5.  $D_{IA}$  and  $D_{IB}$  ( $D_{OA}$  and  $D_{OB}$ ) are inputs (outputs) of tasks A and B, respectively. Type 1 and 2 dependencies are straightforward. Moreover, type 3 is a special case of type 2, while type 6 is a special case of type 5, and type 9 is a special case of type 8. Type 2 and 3 dependencies prevent two tasks from executing concurrently because they compete for the same input data. Types 5 and 6 indicate only one of these tasks will be executed because their outputs overlap and cannot be written concurrently. Types 8 and 9 impose a sequential constraint on the two tasks because one needs the other's output to start. Furthermore, a combination of these relationships can decide the execution order of two tasks. For example, a combination of types 1 and 7 means two tasks can be executed simultaneously; a combination of types 4 and 9 defines a sequential ordering between two tasks, etc.

Table 5: Possible data relationships between two tasks

Type	Relation	Description
1	$D_{IA} \cap D_{IB} = \phi$	Task A and Task B have no common input data.
2	$D_{IA} \cap D_{IB} \neq \phi$	Task A and Task B have common input data.
3	$D_{IA} \supseteq D_{IB}$	Task B uses no more input data than Task A.
4	$D_{OA} \cap D_{OB} = \phi$	Task A and Task B have no common output data.
5	$D_{OA} \cap D_{OB} \neq \phi$	Task A and Task B have common output data.
6	$D_{OA} \subseteq D_{OB}$	Task A produces no more output data than Task B.
7	$D_{OA} \cap D_{IB} = \phi$	Task B does not use Task A's output.
8	$D_{OA} \cap D_{IB} \neq \phi$	Task B uses Task A's (partial) output as input.
9	$D_{OA} \supseteq D_{IB}$	Task B only uses Task A's output data.

It is important to realize that dynamic changes made to a workflow process routing in order to handle exceptions may produce violations in the above data dependencies. The dependency analysis in Table 5 can also be used to improve the design of a process by determining suitable task boundaries. Intuitively, if a task (or activity) uses multiple input resources to produce multiple output resources, then it may suggest it can be divided into two separate tasks (see Figure 4). On the other hand, if two tasks have a sharing dependency on the input side and a fit dependency on the output side (see Figure 5), then we may consider combining them into one task. Detailed discussion of an algorithm to determine suitable boundaries is beyond the scope of this chapter. Further discussion of data dependencies appears in [28, 30].

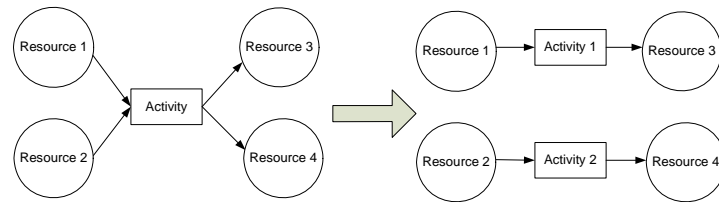


Figure 4 : Split a task based on the dependency analysis

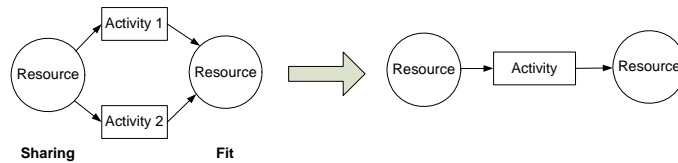


Figure 5: Combine tasks based on the dependency analysis

In summary, the general procedure for creating and running resource-driven workflows is as follows:

- Create a database schema that describes the resource requirements for each task in a workflow.
- Create a schema for each resource to describe the resource and the availability of the resource.
- Run database queries to identify tasks for which all resources are available.
- Perform data dependency analysis and check if guard-in constraints are satisfied.
- Identify a subset of tasks that are executable.

- Execute the subset of tasks; check if guard-out constraints are satisfied; and, update the database.
- Identify a new subset of executable tasks and execute it.
- When all the tasks are executed for the process or its exit conditions are satisfied, the workflow is completed.

More details about architecture and implementation are discussed in the next section. In particular, we shall describe a layered architecture and illustrate how it can be implemented in a database system.

#### **4 A general architecture for a resource-driven workflow**

We propose a four-layer architecture for modeling resource-driven workflow systems as shown in Figure 6. The four layers are *schema*, *runtime*, *scheduling*, and *application layer*. The *schema layer* defines workflow processes, which consist of tasks, instance-level resources and process-level resources. The *runtime layer* specifies how processes and tasks are started and ended. The *scheduling layer* manages assignment of resources to a task so that they can be executed. This may entail use of suitable assignment algorithms that are outside the scope of the current paper. The *application layer* provides links between the workflow system and the applications. It defines how application data can be linked to the corresponding resources. Since there is a clear separation between workflow data and application data, the details of the application data are not important here in the context of the workflow architecture.

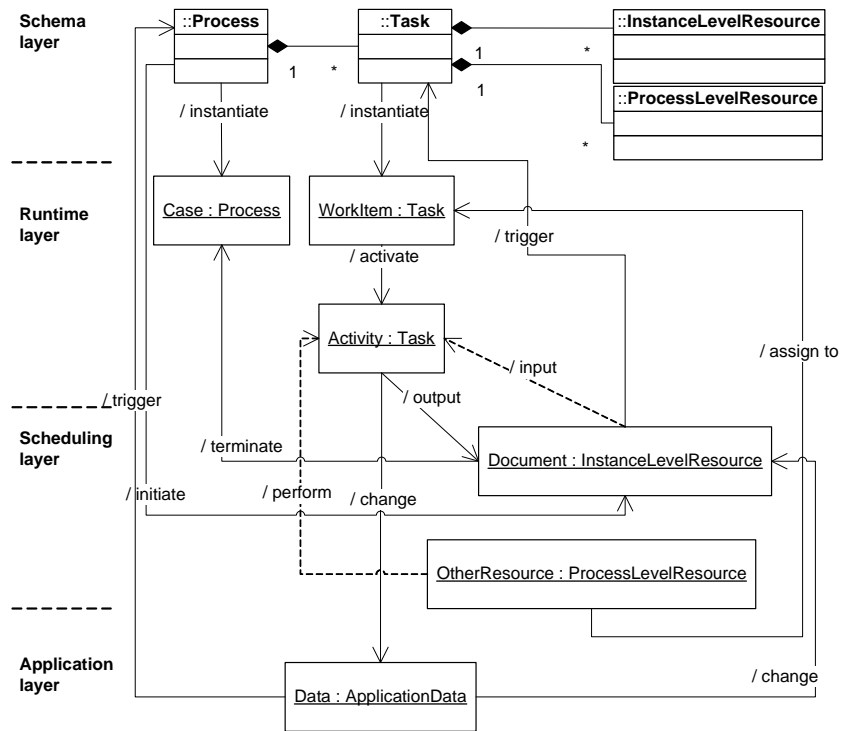


Figure 6: A resource-driven workflow architecture

The significant differences between resource-driven workflow systems and conventional control flow-based workflow systems lie in the runtime and the application layers. In *resource-driven workflow systems*, a process is instantiated into a case when certain instance-level resources (i.e. documents) arrive. In Figure 6, drawn in UML syntax with classes and associations, *Process* and *Task* are top-level classes, and *case* and *workitem*s are their subclasses, respectively. A set of initial documents of the process instance (or *case*) are created as instances of the instance-level resources. Other resources are instantiated similarly from the *ProcessLevelResource* class. A *task* is instantiated into a *workitem* when its input documents exist. The input documents required by one task are usually the output documents from a previous task, except the initial documents for the first task, which are generated by the process repository when the process is instantiated. After a *workitem* gets its input documents and associated resources (at the scheduling layer), it becomes an *activity* that can be executed. An activity potentially changes the values in its input documents or produces new documents, thus making next tasks ready to run. The dotted lines in Figure 6 show that an activity uses input document data and needs other resources to perform a task. A case terminates when documents satisfying its exit conditions are produced.

#### 4.1 A prototype for a resource-driven workflow system.

We have implemented an initial prototype system using Transact-SQL on a Microsoft SQL Server 2000 [22]. Triggers are used to enact the workflow system. The framework presented in Figure 6 is mapped into a DBMS using the execution architecture described in Figure 7. It shows that when a database table is changed (through an insert, update, or delete operation), a corresponding trigger is fired. This trigger generates appropriate events and puts them into the event queue table. Then the trigger associated with the event queue table sends new event messages to event listeners and the listeners execute all the event adapters who registered for these events. Finally, the event adapters update the associated tables and start the next iteration. The architecture shown in Figure 7 consists of two loops: *workflow layer loop* and *application layer loop*. The workflow layer loop updates the workflow tables (through Workflow EventAdapters) and the application layer loop updates the application table (through Application Event Adapters). An event adapter propagates information about an event to the appropriate database tables. There are two types of triggers shown in Figure 7, the system triggers (i.e. workflow and event triggers) and application triggers. Note that Figures 6 and 7 offer two different perspectives, and the layers in the high level design architecture of Figure 6 map only approximately into the execution architecture of Figure 7. The top two layers in Figure 6 are captured in the workflow and application tables in Figure 7. The scheduling and application layers in Figure 6 are incorporated into the workflow and application event adapters, respectively, in Figure 7.

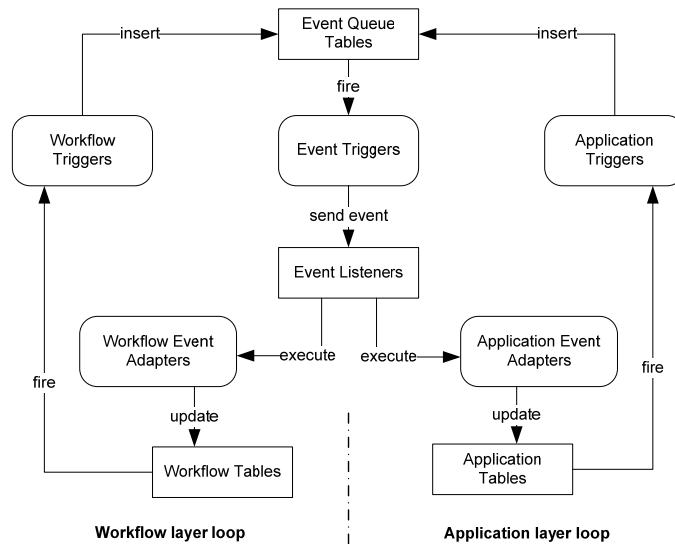


Figure 7: Execution architecture for a resource-driven workflow system

Another advantage of implementing a workflow system inside a database is that transaction management features, such as concurrency control and crash recovery, are already built into most database systems. In the next section we show how exceptions are handled in a resource-driven workflow system using these features of the underlying database.

## 4.2 Handling exceptions by task deferral

We show here how resource-driven workflows facilitate easier handling of exceptions. The main idea is that when a task throws an exception it may be skipped or deferred. To skip a task one can simply remove it from the list of required tasks and make sure that no other task depends on the output from this task. *Task deferral* is more sophisticated because several issues must be considered before deferring a task. First, a task should be deferred only if some alternate task can be started instead of it. If the output of the task is required by all the succeeding tasks, it cannot be deferred without redefining the subsequent tasks. Second, deferring a task changes the workflow execution path, which may cause complex process changes and even lead to an invalid process. Third, deferring a task increases compensation cost if the delayed task eventually fails.

```
1: Capture exception (by a timeout)
2: Find the task to be deferred
3: Assign a temporary result to the deferred task
4: Repeat{
5:   Get remaining ready tasks
6:   Add to promoted queue
7:   Select a task to run
8:   Run selected task with temporary result from deferred task
9:   if (deferred task has finished)
       if (temporary result == actual result)
           {exit and resume normal processing}
       else
           {roll back promoted tasks}
       } until (no task is ready)
10: At end if deferred task is still running, then abort.
```

Figure 8: Procedure for task deferral to handle exceptions

Task deferral is achieved by relaxing soft constraints. Thus, we may relax soft constraint 1 of Figure 2(b) if, say, the credit approval system is temporarily down, or soft constraint 2 if the invoicing system is unavailable. In such cases an exception may be thrown, perhaps when a deadline for a task completion expires and a

timeout occurs. Figure 8 gives a proposed algorithm for handling a task deferral. In this algorithm the states of a task are: *ready (to run)*, *running*, *deferred* and *finished*. A task (or workitem) is ready to run when all its input resources are available, and it becomes an activity at this point. When a task throws an exception, the workflow runtime environment first captures the exception (step 1), identifies the task to be deferred (step 2) and assigns a temporary value to its result such as "done"/"fail" or "true"/"false" (step 3). The temporary values may be assigned based on rules or past frequencies. Then, it identifies those that can be executed without completion of the deferred task (step 5) based on the temporary value assigned in step 3. This is done by searching all the tasks for which the input resources are already available. These tasks are called *promotable* and in step 6 they are added to the *Promoted Queue*. Then in step 7, the workflow runtime environment performs a dependency analysis on the set  $P$  of promotable tasks and finds the one which does not depend upon any other task in  $P$ . This task is executed in step 8. Upon completion of this task, the workflow system will look for the next task to be executed based on the new state of documents. The deferred task may either have finished or it may still be running (step 9). If the deferred task is still running, steps 4 through 9 are repeated to find yet another promotable task. On the other hand, if the deferred task has finished, then the normal processing can resume. However, we need to compare the temporary result assigned to the deferred task (in step 3) with the actual result. If they are different then the promoted tasks must be rolled back. The schema should specify whether a task can be rolled back. Finally, if a deferred task is still running at the end of the process or after a maximum time limit, then it must be aborted and the entire process is rolled back. Roll back can be done quite easily by treating the entire process as a transaction, and then the promoted tasks as a sub-transaction within the outer transaction. Then it is possible to use the SQL Rollback statement with appropriate parameters to rescind all changes of the current transaction.

The above procedure can be easily implemented when there is a soft constraint between a deferred task and its successor. By assuming a temporary output from the deferred task, the workflow instance can proceed. The temporary output can be included as part of the workflow schema. Obviously, this is only possible for planned exceptions. In the case of unplanned exceptions, this approach would require that a temporary value be provided by the user for the output of the deferred task. The major advantage of using temporary outputs for deferred tasks lies in the simplicity of the approach. It doesn't violate the dependencies, so the correctness is guaranteed. However, it may cause extra compensation cost if the actual output of the deferred task cannot be easily predicted. Therefore, this approach is suitable only when the output of the deferred task can be predicted with a high probability, e.g. an assumption like, most credit card transactions are approved.

Table 6: Handling of hard and soft constraints between two tasks, A and B

<b>Constraint Type</b>	<b>Description</b>	<b>Example (see Figure 2(b))</b>	<b>Handling</b>
Hard	Output of Task A is input for Task B	check credit can only be done after order is received	Task B can start only after Task A is finished
Soft constraint 1	Guard condition for Task B	check_credit.status=="done"	Relax constraint and start Task B. Later, check actual status value
Soft constraint 2	Guard condition for Task B	Invoice.status=="done"	

Table 6 summarizes the main scenarios for our approach. By relaxing certain soft constraints that serve as guard conditions for a task, it is possible to proceed past the delayed or deferred task temporarily. However, this does not mean that we are skipping this task altogether. As noted above, before the instance is completed, a check must be made to ensure that the task did finish, and the actual result was indeed the same as the one presumed; else, the subsequent tasks are rolled back. Finally, if the deferred task is still running when all other tasks have finished or after a certain time limit, then it is aborted and the other tasks are rolled back.

## 5 A comparison of two approaches

A summary comparison between the resource-driven and control flow driven approaches is given in Table 7. Flexibility is an important issue in a workflow management system. Different methods such as structured processes [15], workflow patterns [3], and Petri-Nets [1], offer varying degrees of flexibility. These techniques are based on a control flow described using modeling constructs like splits, forks, joins, and other complex flow structures. On the one hand, some structures like forks enhance parallelism and thus flexibility. But, on the other hand, a predefined control flow also restricts flexibility by forcing a certain ordering of tasks. The resource-driven design can dynamically discover the process flow simply based on the resource dependencies. Thus, if a task generates multiple documents, a subsequent task that needs only the first one can proceed without waiting for the task to finish. Such situations of *partial dependencies* are quite common, and one can increase throughput by exploiting them. In fact, in a real-time workflow the need for an input document may also be deferred in some situations to meet deadlines by presuming temporary default data values from it as explained in the previous section. This can be done easily by relaxing soft constraints (or business rules) and is much harder to do in a control-flow driven approach where it is difficult to distinguish between soft and hard constraints. This added flexibility makes the resource-driven approach especially suitable for *ad hoc* workflows. Lack of flexibility can hinder effective use of workflow systems because actual work practices often differ from predesigned processes and exceptions also arise. This may require changing the order in which certain tasks are done. Our approach can handle such operations relatively easily.

Our approach also relies on the use of database triggers. The use of triggers in workflow system has been discussed in the WIDE project [12] as a way to capture events and handle exceptions in addition to the normal workflow which is designed as a control flow. However, our study takes this approach one step further by using triggers as mechanisms to drive and enact the workflow system, thus obviating the need for a "workflow engine" module. As a result, the workflow system can be implemented entirely inside the database and is more scalable because database systems can handle thousands of transactions per second.

A user does not have to worry about the control flow design, and verification is also easier in our approach. In a control flow driven workflow system, the structure of the control flow must be checked to ensure there are no deadlocks, live-locks, or other problems. In a resource driven workflow, it is only necessary to analyze resource flows between tasks and ensure that each task will obtain its input resources. Such a workflow is also more scalable because database systems can handle thousands of transactions per second, whereas most workflow systems have throughput rates that are much slower. Moreover, resource driven workflows can interoperate with one another more easily if they use common database schemas. In the case of control driven workflows this is harder because there is no accepted standard yet for describing control flows. By far the biggest disadvantage with our approach is that it is harder to visualize the process graphically. In a control flow based workflow this is much easier because the control flow is always depicted visually, and it shows the temporal relationships between various tasks. Of course, one could use the information in a resource driven workflow description and convert it into a control flow, but algorithms for doing so are not discussed here.

Table 7: Comparison between resource-driven workflow and control flow based workflow

Resource-Driven Workflow	Control Flow Driven Workflow
The process is driven by the resources	Process is driven by the predefined control flow
The process is very flexible and can be changed instantly by changing constraints.	The process is less flexible because the limitations imposed by flow patterns are hard to change.
More suited for ad hoc workflows.	Better for production workflows with mature processes
Clear separation of hard /soft constraints	No such separation
Exceptions are easier to handle	Exceptions are not so easy to handle
Verification is relatively easy	Verification could be hard
More scalable as part of a DB system	Less scalable; workflow systems are usually small
Interoperability is easier because resource information is in standard SQL database	Interoperability is harder because different workflow systems use different representations
Difficult to visualize the process	Process can be visualized easily

## 6 Discussion

The main idea behind our proposal is that a *process* is driven by *resources* such as data, human or system roles, physical space and equipment rather than an explicit, predefined *control flow*. A *task* is instantiated into a *workitem* when its input documents exist *and* any associated *guard* constraints are satisfied. After a workitem gets its input documents and other associated *resources* (at the scheduling layer), it becomes an *activity* that can be executed. An activity produces new documents, and changes the database which triggers the next task. The process completes when all tasks are executed. Moreover, soft constraints that reflect business policy can be added separately through guard conditions. This means that when business policy changes only the soft constraints are modified without a need to change a control flow diagram. Constraints have been studied extensively in many database systems and they are usually represented as ECA (Event-Condition-Action) rules [21]. A key aspect of our approach is that it can be executed inside a database, i.e. the database system becomes a workflow engine. Since databases are very fast and more scalable than workflow systems, they can handle larger numbers of workflow instances than workflow engines, thus leading to better performance.

Control-flow-driven workflows are based on basic patterns such as sequence, choice, parallel and loop, and advanced patterns such as multi-choice, interleaved parallel, etc. However, all workflow products don't support all the patterns and this can affect interoperability. In the resource-driven approach, the patterns are not specified explicitly; rather they arise as a result of resource dependencies. Thus, if there is an input-output dependency between two tasks, they are in sequence. If the guard-in conditions of two tasks are in conflict, then they are in choice, and if two tasks have mutually exclusive guard-in conditions and no data dependency between them, then they are run in parallel. A loop involving one or more tasks is created by changing the status of a running task in a workflow instance from 'done' to 'undone'. This would force the tasks to be rerun as in a loop. Advanced patterns can also be simulated by using the guard conditions and locking features of a database. Guard-in conditions can help to select a subset of tasks to execute from a larger set that is potentially executable. A task, while running, may optionally lock a document if it needs exclusive access. If a document is locked by a task, then it must be unlocked before another parallel task can access it, thus creating the effect of interleaved parallel routing.

There are several current approaches based on the control flow perspective of workflow. Examples are Petri-nets [1], XPDL [31], BPEL [24], BPMN [23], etc. These approaches are quite expressive for modeling the control flow, but they do not model resources very well. On the other hand, among approaches that focus on resources, the WIDE project [12] and ADOME-WFMS [7] use ECA rules in RDBMS and OODBMS respectively, which do not explicitly model the control flow. ADEPT takes a more comprehensive approach which includes both data flow and control flow, and is promising for solving most dynamic change prob-

lems [25]. There are other proposals such as Placeless documents project [10], which adds action code into documents, so the coordination can be done within the documents and no explicit workflow system is required. An approach for entity-centric process models is described in [5]. In this approach the main organizing principle for creating processes is entities, which can be treated as a kind of a resource. In the EPC [27] approach each activity has input events that trigger it, and output events that it produces which in turn trigger other activities. This approach has some similarity to our proposal; however, EPC diagrams are essentially control flow diagrams. In a broader sense, it is also noteworthy that despite a plethora of approaches for modeling workflows, there is as yet no established standard that is used widely and can serve as a means to exchange workflow schemas between organizations.

Research on exception handling in workflows is still quite limited. Some work on exceptions in workflows is discussed in [9, 13, 16, 18]. A different perspective for handling exceptions based on deadlines is presented in [4]. WIDE manages exceptions by first activating a local, process specific exception handler, and then allowing propagation of the exception to the parent process. ADOME-WFMS uses Problem Solver Agent (PSA) to handle exceptions.

Another kind of exception can be handled through resource delegation. Thus, if a resource is not available to perform a task that has a tight deadline, then a substitute can be found. For a human resource a subordinate or a superior substitute may be assigned. Similarly, for space and equipment resources, substitutes may be kept in the database and assigned in order to expedite a task if the desired resources are not available. We do not go into details here, but there is related work on delegation in the literature [29].

## 7 Conclusions

This chapter provided a general framework for the design and implementation of resource-driven workflows in contrast to conventional control-flow-driven workflows. In a resource-driven workflow, resources serve as an organizing principle. The tasks in a process are executed in the correct order based on the availability of *resources* such as data documents, human or system roles, physical space and equipment rather than an explicit, predefined *control flow*. We argue that when multiple, dynamic and possibly conflicting resources are involved it is not possible to pre-design a business process based on the control flow alone; rather it emerges from the interaction of resources that are a prerequisite for each task in the process. We showed how resource-driven workflows are especially promising for ad hoc workflow environments, and can be implemented within a database system, thus obviating the need for a workflow engine. A distinction was also made between hard constraints that depend on data dependencies and resource availability, and soft constraints that are determined by business rules. This distinction leads to a systematic way of designing business processes, and also

enables relaxation of soft constraints to handle exceptions. Handling exceptions within a database becomes easier because most databases systems provide roll-back capability.

There are several avenues for more work in this area. First, there is a need for a language to describe resource driven workflows. Second, the types of resources to be modeled, and the level of detail at which each resource is modeled, should be investigated further. Naturally there is a trade-off here between modeling complexity and the value gained from the model, and it should be explored further. Third, algorithms for converting resource-driven workflows into an equivalent control flow for visualization purposes should be developed. Finally, more detailed quantitative comparisons between the resource based and control flow based approaches, perhaps through simulations, would also be helpful.

## References

1. Aalst, W.M.P. van der, The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1): 21-66, 1998.
2. Aalst, W.M.P. van der and Kumar, A., A Reference Model for Team-Enabled Workflow Management Systems. *IEEE Transactions on Knowledge and Data Engineering*, 38: 335-363, 2001.
3. Aalst, W.M.P. van der, *et al.*, Workflow Patterns. *Distributed and Parallel Databases*, 14(1): 5-51, 2003.
4. van der Aalst, W. M., Rosemann, M., and Dumas, M. 2007. Deadline-based escalation in process-aware information systems. *Decis. Support Syst.* 43, 2 (Mar. 2007), 492-511.
5. Bhattacharya, K. *et al.*, Towards Formal Analysis of Artifact-Centric Business Process Models. Proc. of Business Process Management, *5th Intl. Conf. (BPM 2007)*, 288-304, Brisbane, Australia, 2007.
6. Botha, R.A. and Eloff, J.H.P., Access control in document-centric workflow systems--an agent-based approach. *Computers and Security* 20 (6): 525-532, 2001.
7. Chiu, D.K.W., Li, Q., and Karlapalem, K., Web interface-driven cooperative exception handling in ADOME workflow management system, *Web Information Systems Engineering*, 26(2): 93 - 120, 2001.
8. Collis, D. J. and Montgomery, C. A., Competing on Resources: Strategy in the 1990s, *Harvard Business Review*, July-August: 118-128, 1995.
9. Curbera, F. *et al.*, Exception Handling in the BPEL4WS Language, W. van der Aalst, A. ter Hofstede, M. Weske (Eds.), *BPM 2003, LNCS 2678*: 276-290, 2003.
10. Dourish, P. *et al.*, Extending document management systems with user-specific active properties. *ACM Trans. Inf. Syst.* 18(2): 140-170, 2000.

11. Dumas, M., Aalst, W.M.P. van der, and Hofstede, A. H. M. ter, *Process Aware Information Systems*, Wiley, 2005.
12. Grefen, P. *et al.*, *Database Support for Workflow management—The WIDE Project*, Kluwer Academic Publishers, 1999.
13. Hwang, S.-Y. and Tang, J., Consulting past exceptions to facilitate workflow exception handling. *Decision Support Systems*, 37(1): 49-69, 2004.
14. Krishnan, R., Munaga, L., and Karlapalem, K., XDoC-WFMS: A Framework for Document Centric Workflow Management System, *LNCS 2465*, 348–362, 2002.
15. Kiepuszewski, B., Hofstede, A.H.M, and Bussler, C. On Structured Workflow Modeling. In Proceedings *CAiSE'2000*, LNCS 1797, Springer Verlag, 2000.
16. Klein, M. and Dellarocas, C. A Knowledge-Based Approach to Handling Exceptions in Workflow Systems. *Computer Supported Cooperative Work (CSCW)*, 399-412, 2000.
17. LaMarca, A. *et al.*, Taking the Work out of Workflow: Mechanisms for Document-Centered Collaboration, *ECSCW'99*: 1-20, 1999.
18. Luo, Z. *et al.*, Exception handling in workflow systems, *Applied Intelligence: the International Journal of AI, Neural Networks*, 13(2): 125-147, 2000.
19. Malone, T.W. and Crowston, K., The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1), 1994.
20. Mazumdar, S. and AbuSafiya, M., A Document-Centric Approach to Business Process Management. In Proc. *Intl. Conf. on Information and Knowledge Engineering*, 461–466, 2004.
21. McCarthy, D.R. and Dayal, U., The Architecture of an Active Database System. In Proc. *ACM SIGMOD Conf. on Management of Data*, Portland, 215-224, 1989.
22. Microsoft Corporation, *SQL Server Books Online: Transact-SQL Reference*. 2000.
23. Object Management Group (OMG). Business Process Management Notation, <http://www.bpmn.org/>.
24. OASIS, “Web Services Business Process Execution Language (WSBPEL)”, <http://www.oasis-open.org>.
25. Rinderle, S., Reichert, M., and Dadam, P., Correctness criteria for dynamic changes in workflow systems—a survey, *IEEE Transactions on Knowledge and Data Engineering*, 50(1): 9-34, 2004.
26. Scheer, A.W., *ARIS – Business Process Frameworks*, 2ed, Springer, 1998.
27. Scheer, A.W., Thomas, O. and Adam, O., Process Modeling Using Event-Driven Process Chains. In *Process Aware Information Systems*, M. Dumas, et al. (eds), Wiley, 2005.
28. Sun, S. X. *et al.*, Formulating the Data-Flow Perspective for Business Process Management, *Information Systems Research*, Vol. 17, No. 4, December 2006, pp. 374-391
29. Wainer, J., Kumar, A., and Barthelmess, P., DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.* 32(3): 365-384, 2007.

30. Wang, J. and Kumar, A., A Framework for Document-Driven Workflow Systems, W.M.P. van der Aalst et al. (Eds.), *BPM 2005, LNCS 3649*: 285-301, 2005.
31. Workflow Management Coalition (WFMC), XML Processing Description Language (XPDL). <http://www.wfmc.org>.