Considerations on Using Transactions in Workflow Design

VICTORIA IORDAN, ALEXANDRU CICORTAS
Department of Computer Science
The West University of Timisoara
Blvd. V. Parvan 4, Timisoara 300223
ROMANIA
iordan@info.uvt.ro, cico@info.uvt.ro

Abstract: - Using the results that were expressed in the formalisms in [19] and our works [5], the paper intend to present some considerations on the workflow and workflow design under a large context: transactional systems, transactional workflows and web services. Our previous work dialed with a multi-agent system for reengineering failed workflows.

Key-Words: - transactional workflow, workflow modeling, multi-agent systems, roles, complex applications design.

1 Introduction
The systems became more and more complex, many of them use transactions and many are transactional. These systems are usually managed using workflows. The resources are limited in amounts and for them the system components and their appropriate workflows compete in order to accomplish the system functionalities.

Using basic works and our works, we will present some considerations on the requirements for workflow design and also for workflow engine design, especially concerning the transactional workflows and resources. Using Enterprise Application Integration technologies companies share their information in distributed information systems and try to automate their Workflow Management Systems (WFMSs), with agents that allow it. For implementing B2B integration many standards and technologies are used, XPDL (XML Process Definition Language), Wf-XML, WSCI (Web Service Choreography Interface) and many derivatives of the BPEL. Business Process Integration was considered as one of the most important techniques for e-business application integration or B2B integration [21]. In [14] was proposed a model for design and implementation methods for B2B workflow integration.

Another problem is what should be considered for workflow-based process integration and how can be adopted the recent standard technology: services and business process languages, in implementing process integration. Usually the implementation during the process integration contains three main interfaces: Application Interface, Human Interface and B2B Interface, in a multi-agent context.

The Web Services and performance of a composite service is one of the focuses of concern [17], [18].

The paper is organized as follows. The second section presents process and its components. The third section treats the complex systems for the workflow interaction by the human person. In the fourth section the atomicity of an activity in the context theory of concurrency control and recovery is shortly commented. The next two sections present the transaction processing and the mechanisms in transaction model processing. In the Fifth Section the details concerning the transaction processing are giving. The Sixth Section deals with the transaction mechanisms in transactional model. The Seventh Section makes some considerations on the web services composition and resources. The Eight Section comments our previous works concerning the reengineering the failed workflows using a multi-agent system. The last Section concludes with considerations regarding the resources, workflow engines and transactional systems.

2 Process and its Components
The processes have many instances that are executed concurrently in the real environment. The Figure 1 illustrates it based on the meta-model defined in [23].

Fig. 1. Process instance state transitions meta-model

The activities that compose the process/workflow instances can evolve, reaching various states and change...
each other during process instance execution as is illustrated in Figure 2 [23].

![Activity instance state transitions meta-model](image)

**Fig. 2. Activity instance state transitions meta-model**

### 3 Roles and Human Interaction

In the context of complex systems the workflow interaction by the human person is usually done by roles. A workflow participant can gain one of these roles according to the process context:

- **initiator** – the person who can create the new task and also can accept or decline its executions; initiator also is capable of terminating the task if need be;
- **task manager** – decides, if the assigned task executor is chosen properly for the current task and also can re-assign the user;
- **executor** – is responsible for the task execution and is directly accountable to the initiator of the task.

The meta-model of workflow participants, roles, functions and resources interaction is defined by WfMC in the Figure 3.

Using this model, the description of the organizational model can be done, where organization is defined as hierarchically accountable departmental structure. Each of departments has its own manager, which can gain initiator or task manager role. The rest of the department personnel can gain task manager or executor roles.

![Organizational structure, used for actor roles identification](image)

**Fig. 3. Organizational structure, used for actor roles identification [23].**

### 4 Concurrency control and recovery

The unified theory of concurrency control and recovery integrates atomicity and isolation in a common framework. In [19] was extended the unified theory by applying it to generalized process structures. The authors [19] claim that their goal is to provide a more flexible handling of concurrent processes while allowing as much as much parallelism as possible. Also in [19] was provided a correctness criterion for transactional processes and were outlined the differences that the structure of transactional processes and traditional transactions. In this work was: proposed a solution for concurrency control and recovery without imposing restrictions on the environment; using the correctness of a single process based on flexible transactions it provides a correctness criterion for concurrent execution of several processes by generalizing and adapting the unified theory of concurrency control and recovery to transactional processes (extending the applicability of models).

The atomic activities are a concept that must be carefully analyzed. Due to the fact that in a process some activities are in a dependency relation in the case of failure of many from these activities in our opinion the compensation must be clearly designed subsumed on the dependency or modify the requirements concerning the environment that is different compared with that from [19]. The retriable activities must be taken into the account in a strong connection with their recovery. Concerning the compensation of an activity we have some remarks. Due to the fact that when an activity $a_i$ fails and it can be compensated by a compensating sequence denoted by $(a_i)^{-1}$ in definition 2 from [19]. Concerning the compensability and compensation (Definition 2 from [19]), we propose: an activity $a_i$ is compensable if there exists a sequence such as the activity sequence $a_i$ is effect-free. The sequence is the compensating activity set of $a_i$.

We consider that for an activity there is not a unique activity but a sequence, because the compensation can introduce many activities that usually do not are real atomic activities in the process these are used only in the care of compensation. And also we outline that in the sequence there are many atomic activities that are used in compensation many sequences, and it is one of the main our concerns that imposed the details that we proposed. Concerning the retriable activities our view is different form that from [19]. One different view is based on the fact that reinvoking many times an activity it can have some distortion on the environment that must be taken into the account. We suppose that an activity is composed from many (atomic) actions (steps) that are executed. An activity can not be invoked many times due to the fact that in the system there are components that are or can be affected by the execution of this...
activity (like an updating of an entity in a data base). If the activity fails and before its failure the part of activity (many actions of the activity) can affect the real system (as an example a data base entity was updated) by invoking this activity many times, some action from it will affect the environment. In this case must take into the account a procedure for recovery. An interesting definition of an atomic activity is given in [25]. The activity of a group of components or objects constitutes an atomic action if there are no interactions between that group and the rest of the system for the duration of the activity. To guarantee the property of compensability, a compensating activity is not itself compensable but it is serial and is guaranteed to commit. In [19] was formalized also the commutative property of activities that is the basis for reducibility that allow to define a criterion for concurrency control and recovery. The basic idea consist in that to eliminate both an activity and its compensating activity (set of compensating activities in our opinion) if it constitutes an effect-free activity sequence. Another main aspect is that of the discarding of cyclic conflicts that can be induced by using the compensating sequences. The workflow designer is charged to solve it. The serializable is used in order to solve the correct concurrency recoverability in [19]. Based on the previous remarks we conclude the following: the atomicity of an activity must be treated in the context of processes with some caution; the compensation of an activity is not an activity but a sequence of activities; in the case of re-invoking a retriavable activity it must be replaced by some recovery actions.

5 Transaction Processing

A transaction is constituted from a series of operations performed as a single unit of work that either completely succeeds, or fails with all partially completed work being undone [13]. Grouping a set of operations in a transaction, the consistency and reliability of the system must be assured despite of any errors that occur. The transaction ends normally when all the operations in it, must complete successfully. The transaction is executed by an application in the context of a system [13], [17]. A transaction ends when either the transaction is committed by the application or the transaction is rolled back by application or system failure, or when a commit operation fails (i.e., due to inadequate resources or data violations). If the transaction successfully commits, changes associated to it are saved and are made visible to new transactions. If the transaction is rolled back, all changes made by it will be discarded (as if the transaction never happened at all). Figure 4 illustrates a state transaction diagram for a transaction.

Fig. 4. The state transaction diagram for a transaction

The transaction has the following intermediary states: active, partially committed. Here the recovery protocols need to be tested in order to ensure that a system failure will not result in an inability to record the changes of the transaction permanently. If any of these checks fail then the transaction transits to the failed state, committed, failed, aborted. The basic properties of a transaction are the ACID properties: Atomicity, Consistency, Isolation and Durability [17]. Usually the environments for complex systems are distributed. The decomposition in atomic elements allows constructing reliable distributed systems where the failures occur. As mechanisms used are the recoverability and serializable. As requirements that are manifest from previous considerations the concurrency control and processing of the distributed transactions in an adequate manner in order to implement adequate mechanisms.

6 Mechanisms in transaction model processing

The transaction model is able to manage relaxed transactional properties and rollback through compensation [10]. Relaxed transactional properties are required for long-living, co-operative processes like workflows to avoid complete undo of performed work and to facilitate sharing of intermediate results of processes. Compensation allows rollbacks in relaxed transactional processes. A transaction model can be used in general process structures including cycles and including the notion of partial compensation through the use of safe points. Support for cycles is an important feature, as many practical workflow applications contain cyclical process structures (e.g., to obtain a business goal in an iterative way or to retry a specific business function). Partial compensation is important in practice to have a flexible means to control the scope of a process rollback. The exception handling is orthogonal to transaction handling [22]. An example of coordinated exception recovery is given in [25] is presented in figure 5.
Here two threads entering compensation action synchronously through different entry points having a common goal. An exception is raised and one of the threads is informed of it, then both threads transfer control to their respective handlers H1 and H2 which attempt to perform forward recovery. External object are put into correct states so that compensation action is able to complete successfully. To obtain relaxed atomicity [10], rollback operations in a transaction model should have application-specific semantics instead of the database-oriented semantics of the local transaction model. Also there was proposed system architecture designed to support the transaction mechanisms.

In [10] was proposed system architecture designed to support the transaction mechanisms and is proposed how the architecture can be extended to deal with distributed global transactions that span multiple process engines. Compensation is considered a proper way to handle application rollback, not only in workflow contexts. Compensation is used for rollbacks, and as an ingredient to higher-level notions. Also, it can be used in a flexible exception handling approach [11]. In [10] is provided a complete formal specification of compensating transaction management mechanisms.

Coordinated exception handling is defined based on cross-organization process interaction, which can be one-way or two-way interaction. Traditional workflow management systems already have some necessary handlers for simple exception handling, usually it is predicted exceptions handling. The handlers are: ignore, record, retry, compensation, alternative task, backwards recovery, forward recovery, consistent state, resumed and processed, propagation, termination, suspension and resumption, procedural exception handler.

### 7 Web Services Composition and Resource Allocation

Composition is one key service design principle under SOA (Service Oriented Architecture). With composition, new complicated composite services can be obtained by aggregating component services together in a workflow. The goal of resource allocation for services concerns all services in the workflow to assure performance targets. Due to the dynamic nature of service invocation, resource allocation and QoS provisioning of services are much more difficult than that of traditional environment. This dynamism is given by: (1) non-deterministic arrival of service requests, (2) non-constant demand for resource capacity of requests (due to different inputs and service level agreements) and (3) execution uncertainty within a service workflow that results in only some component services inside to be used.

The dynamism results in the need for runtime adjustment of resources allocated to individual component services. As one possible solution is to improve the accuracy of future workload prediction of individual component services of a workflow dynamically, and do the proper adjustment of resource allocation based on this prediction in advance of the actual workload arrival [24]. SOAP, WSDL, and UDDI [6] aim at providing infrastructure to support Web service composition. BPEL4WS (Business Process Execution Language for Web services) [7] combines Microsoft XLANG [20] and IBM WSFL (Web Service Flow Language) [15] to provide language support for the formal specification of business processes and business interaction protocols. WSCI (Web service Choreography Interface) [2] is an XML-based interface description language that describes the flow of messages exchanged by Web services. BPML (Business Process Modeling Language) [3] is designed to express abstract and executable processes that address all aspects of enterprise business processes. Also are used for service description and composition include ebXML [18] and DAML-S [1]. Many works propose the service modeling and frameworks, but as a major problem in our opinion remains that of the resource allocation.

Traditional resource allocation solves the problem of limited resource availability when processes try to access resources. Some processes might be able to access resource concurrently, while other processes might require exclusive or sequential access to the resource [16]. In distributed systems, how to allocate resources to tasks determines the system efficiency, throughput [12] or resource utilization [9]. However, their workloads are relatively fixed and static, the resource allocating methods are static as well, and they are not customized to previous workflow with multiple component services. In [24] was proposed a dynamic resource allocation scheme that uses the given service transition probabilities, which estimates the future workload of each component service and finally allocate resources to them accordingly. The dynamic resource allocation scheme for composite service is given in four
steps: (1) Computation of transition probability and estimation of request arrival time. (2) Real time dynamic service workload prediction.(3) Generation of service workload prediction result. (4) Dynamic service workload adjustment. For a given service the allocation (including service replication, if necessary) must be done before the actual workload arrival. As a principle, one of the most usual is that of the just in time. The prediction result will be more accurate if the prediction point service is closer to the predicted service. If the replication time is taken into consideration must taken into the consideration other features.

Based on the previous, concerning the transactional workflow, we can conclude that: the transactional semantics are an important aspect of workflow management, transaction mechanisms dedicated for workflow environments. The global transactions use compensation to perform rollback operations and the forward rollback mechanism can be used after compensation and it provides as alternative strategies: redo, alternate path and terminate.

8 Our previous works

Workflows and especially those for business processes are generally in a very high dynamic. So, during the execution the state of the system and also of the workflow can change, or can fail. Whereas the system state is stable the workflow can evolve but while in some state the system changes in an unpredictable way the workflow will encounter particular cases (states) that were not developed. In such cases the workflow must be able to adapt to the new conditions. During the execution of an activity, of a workflow instance, a failure of the activity can appear. In this case the workflow will be stopped and its reply can be done, in the actual circumstances, or can be done only from its beginning.

In a workflow there are activities whose result is in one of the following categories [5]: the activity result can be retained and in future can be used, we can say it is persistent; the activity result is volatile in future, it does not be used, if we want the activity result in the future, we must reply it; the activity result has affected the real system state and the system state must be restored in the case if we want to replay this activity we must remove this effect.

If the workflow fails, then it fails due some activities that it failed. The previous workflow activities were executed and based on their result as previous analysis was done, we can find the activities whose results are reusable and are persistent. In the case of the reply of the workflow we must make an analysis that: restores system state due to the activities that were affected in some way the system state, find the activities that have reusable and persistent results, reconstruct a new workflow with the remained activities and reusable results that were obtained till the failure appeared. (The problem is that on every parallel branch we must have a reusable result).

Based on the previous we proposed a multi-agent system that uses an extended data base for workflow instances. The activities stored in the data base must have some specific attributes in order to allow to the workflow engine to use them in the failure of a workflow instance. The details can be found in [5] and the model is given in the figure 6.

9 Final considerations

Usually in complex systems there is a need for multiple applications to access the same data base at the same time. These applications may interfere with each other and cause the data base become inconsistent. It requires that the transactions in these systems preserve individually the consistency of data base. Whenever two (or more) transactions are executed concurrently their data base operations interleave, i.e., operations from one transaction may execute in between operations from another transaction and thus interfere with them. This interleaving can produce the corruption of the consistency of data (the data base can be in an inconsistent state). That requires that we must implement a concurrency control that has as main objective to not allow this kind of interference and avoid potential errors. As requirements that are manifest from previous considerations, the design of the concurrency control and processing the distributed transactions in an adequate manner in order to implement adequate mechanisms.
For the transactional workflows the requirements concerning the workflow engine are more complex due to the fact the transactions have their specificities that must be accomplished. Based on the above requirements and our previous work, [5], we propose that the workflow, its instances and their component must be represented specific data bases. During the instance execution for every activity/task it results of execution is stored in the data base. In the case of a failure of a task its instance will be reengineered and it will restart from the last consistent state.

References: