Distributed Global Transaction Support for
Workflow Management Applications♣

Jochem Vonk, Paul Grefen, Erik Boertjes, Peter Apers

Center for Telematics and Information Technology (CTIT)
University of Twente
{vonk, grefen, boertjes, apers}@cs.utwente.nl

Workflow management systems require advanced transaction support to cope with their inherently long-running processes. The recent trend to distribute workflow executions requires an even more advanced transaction support system that is able to handle distribution. This paper presents a model as well as an architecture to provide distributed advanced transaction support. Characteristic of the transaction support system is the ability to deal with arbitrary distribution of business processes over multiple workflow management systems and the support for flexible rollbacks. The modularity of the architecture and the orthogonality with respect to the workflow management system allows the transaction system to be applied in other application areas as well. The high scalability of the architecture allows an arbitrary combination of transaction support systems and workflow management systems of which the locations are irrelevant. In the WIDE project, the developed technology is applied to the commercial FORO workflow management system.

1 Introduction

As indicated by recent studies [1, 2, 17, 5], transaction support is a necessary functionality required by workflow management systems (WFMSs). However, the traditional ACID transaction paradigm is too strict for the inherently long-running processes of WFMSs. Because of the characteristics of WFMSs a relaxed notion of atomicity and isolation is required [15, 12, 5].

Another functionality required for WFMSs is the support for distributed workflow executions. Large organizations are usually divided into multiple business units, each operating their own WFMS. However, the overall business processes concerning the business goals of the complete organization are performed by some or all of the separate business units, i.e., the business processes are distributed over the multiple business units of the organization. WFMSs should therefore support the distributed

♣ The work presented in this paper is supported by the European Commission in the WIDE Project (ESPRIT No. 20280). Partners in WIDE are Sema Group sae and Hospital General de Manresa in Spain, Politecnico di Milano in Italy, ING Bank and University of Twente in the Netherlands.

© Springer-Verlag Berlin Heidelberg 1999
execution of the workflow processes. This requires the transaction support system to cope with the distributed execution as well.

In the WIDE project, a two-layer transaction management system has been designed [12, 14]. The upper layer caters for the requirements of long-running processes by offering global transactions with relaxed ACID properties and is based on the saga model [11]. A global transaction is divided in steps whose actions are committed to the database after the step finishes. Rolling back already finished steps relies on compensating actions that semantically undo the effects of completed actions. In the WIDE project, the saga model has been extended to support flexible rollbacks. The lower layer of the transaction model is based on the nested transaction model [18], which provides the more strict traditional ACID properties required by the individual global transaction steps. It is completely orthogonal to the upper layer and is thus not influenced by the extensions made to the upper layer. The lower layer is not relevant in this paper and the reader is referred to [4].

This paper discusses the extension to the WIDE upper layer providing support for the distributed execution of workflow processes, i.e. distributed global transactions. The developed distributed transaction model offers different abort modes to allow for flexible rollbacks. It also offers a mechanism to handle concurrent aborts. The architecture of the non-distributed global transaction support system [14] is extended with a communication protocol to coordinate the different systems. The architecture allows for an arbitrary distribution of transaction support systems and WFMSs.

The paper is structured as follows. Section 2 describes related work. Section 3 discusses distribution of workflow executions. The distributed global transaction support is described in Section 4, of which the architecture is presented in Section 5 together with implementation issues. Section 6 presents concluding remarks.

2 Related Work

Numerous advanced transaction models have been proposed in the past [9, 17, Chic98]. The aim of the WIDE project is not to propose yet another advanced transaction model, but to reuse what is already available. A combination of two models, each supporting different requirements of workflow management systems, is chosen as a basis and both are extended to provide for more advanced features. This paper discusses the extensions made to upper layer of the WIDE transaction with respect to distributed workflow executions and flexible rollbacks.

The transaction model used in the Exotica project [1] is also based on the saga model, but relies on statically computed compensation patterns. The process structure supported by Exotica is more limited than the one supported in WIDE, e.g. cycles are not allowed. Consequently, its functionality is limited compared to the work presented in this paper.

A transaction model similar to ours is discussed in [8]. Different failure sources and failure classes are analyzed. However, the distributed execution of workflows and the required transactional support for it is not covered.

Distributed workflow execution support is described in [3] by introducing Information Carriers (INCAs) All information necessary (data, control flow, etc.) to execute a workflow is contained in INCAs, so there is no real notion of a WFMS. INCAs are passed to the autonomous processing stations involved in the distributed
execution. The transactional functionality offered depends on the transactional functionality offered by the processing stations. If a processing station provides the ACID properties, compensating steps are used to undo the completed steps. Which compensating steps need to be executed is specified by rules in the INCAs.

In the Mentor project [23], distributed workflow execution is addressed. A transaction processing monitor (Tuxedo) is used to provide for failure tolerance and reliable message passing. Transactions in Mentor are more restrictive and comply to the strict ACID paradigm, while the model presented in this paper allows for relaxed ACID properties.

The Workflow Management Coalition has specified a standard interface to facilitate the interoperability between different WFMSs [22] allowing for distributed workflow execution, even in a heterogeneous WFMS environment. Transactional issues are not addressed, except for writing an audit log. The work presented in this paper can thus be seen as complementary to the standard interoperability interface.

Commercial workflow products, like Cosa and Staffware, do not offer much transaction support [6, 21]. For example, Cosa treats every workflow activity as a separate ACID transaction.

3 Distributed Workflows

Large organizations are usually divided in multiple business units which have to work together to perform the business processes that involve the entire organization. The WFMSs in use by the separate business units need to interoperate to support the overall business processes, i.e. those business processes that are distributed throughout the organization.

The distribution of a process over multiple business units can be seen as the delegation of parts of the process, called subprocesses, from one business unit to another business unit. For example, a billing department performs some actions (the subprocess) for the purchase department. A delegation model is therefore adopted for the distribution of processes over the different business units, and thus over the different WFMSs. In the delegation model, WFMSs can delegate the execution of subprocesses to other WFMSs. This implies that the distributed workflow execution topology is tree-shaped, i.e., one WFMS can delegate to multiple other WFMSs, but a delegated subprocess cannot originate from more than one other WFMS, see Figure 1a in which the dotted lines represent site boundaries. Note that the workflow execution topology might be different than the WFMS topology. Although the
execution topology is tree-shaped, the WFMS topology might not be, e.g. in the case where one WFMS executes multiple delegated subprocesses. An example of a distributed workflow process specification is shown in Figure 1b which represents the workflow process “selling a holiday trip to a customer” of a large travel agency. The travel agency consists of several business units that are involved in the process, the customer buys the holiday trip at the sales office, the financial department handles the invoices, and the booking department arranges the travel documents and vouchers. All involved business units have to perform their own subprocess using their own WFMS. The subprocesses together form the complete “selling trip” process.

In the figure, each distributed subprocess is represented by a global transaction which is a graph consisting of circles and solid arrows. The circles represent actions, called global transaction steps, and the solid arrows indicate the order in which the global transaction steps can be performed. The circle marked with ‘SP’ is a global savepoint, which is explained in the next section. The dotted arrows indicate the delegation of a subprocess to another WFMS.

Figure 2a: Distributed WFMSs and global transaction systems

As each separate subprocess is a global transaction, executed by a certain WFMS, it can per site be supported by the non-distributed global transaction support (GTS) system as described in Section 1, see Figure 2a. The connection of the subprocesses, represented by the dotted arrows, implies the order in which the subprocesses are executed. The connected global transactions can therefore be seen as one large distributed global transaction. Only the distribution aspect requires an extension to the global transaction support system to handle the delegation of subprocesses. This extension is described in the next section.

4 Distributed Global Transaction Support

As described before, the upper layer of the transaction model developed in the WIDE project relies on compensating actions to semantically undo the effects of completed global transaction steps in case of failures. Compensating an entire workflow, thereby undoing all the work that has been done so far is called a complete abort. A complete abort is usually too strict and undoing only part of the process would suffice. For this purpose the global savepoint concept is introduced to allow for flexible, partial aborts. When a partial abort is initiated because of some failure in the workflow execution,
the executed workflow is only rolled back (compensated) to one or more savepoint(s), thereby avoiding rolling back the entire workflow execution [13]. The part of a workflow that needs to be rolled back is dynamically calculated by the global transaction support.

In the distributed workflow execution scenario, as presented in Figure 2a, every WFMS is supported by a global transaction system, which can handle aborts of global transactions for that specific site. This means that the extension to the global transaction system to cope with distribution does not change the algorithms to calculate a compensating global transaction for one site, i.e. in a non-distributed scenario. In the distributed scenario however, an additional communication protocol is required that decides which other sites need to abort as well and in which abort mode (partial or complete abort). It then signals those other sites to actually start an abort in the correct abort mode.

4.1 Backward and Forward Aborts

To illustrate the way aborts are propagated through the workflow execution, an example is presented in Figure 2b. In the figure, the black circle at site 5 is the workflow activity that fails, thereby causing an abort. Suppose the abort request should be performed in partial abort mode. The transaction system at site 5 finds no savepoint at that site to roll back to. It signals the parent site to abort, called backward abort, in partial abort mode. The subprocess aborted at the parent site can roll back to a savepoint so no more backward aborts are necessary. However, since the part of the subprocess that is rolled back has delegated a subprocess to site 4, that subprocess must be aborted as well. The transaction system will therefore signal the child site to abort, called forward abort. A forward abort implies that the site receiving the forward abort request must abort in complete abort mode as all the actions done at that site are dependent on an action that has been rolled back and are therefore invalid.

In summary, an abort involves multiple sites if the following situations occur:

• A complete abort request is issued by a WFMS that is executing a delegated subprocess; at least one backward abort is issued.
• A complete abort request is issued by a WFMS that has delegated a subprocess; at least one forward abort is issued.
• A partial abort request is issued by a WFMS that has delegated a subprocess and for which the global transaction step that delegated a subprocess has to be compensated; at least one forward abort is issued.
• A partial abort request is issued by a WFMS for which there is no savepoint in the subprocess to roll back to on that site, see example above; at least one backward abort is issued.

If the global transaction support system has computed that other sites need to abort as well, the communication protocol will signal those sites to abort in the abort mode that depends on the following two rules:

1. The parent site: backward abort in the abort mode that was issued by the failing site
2. The child site: forward abort in complete abort mode.

As each WFMS might independently decide that it needs to abort, concurrent aborts can occur. If those aborts trigger forward and/or backward aborts, the concurrent
aborts might come together at one site, causing an abort conflict. The way this is handled is described in the next section.

4.2 Concurrent Aborts

Inherent to distributed workflow execution is the possibility that different subprocesses concurrently issue an abort request. Because of the backward and forward aborts, which might follow an abort request, the occurrence of concurrent aborts might lead to abort conflicts. Abort conflicts occur when one site must handle different abort requests, e.g. a subprocess at one site receives a partial abort request from one child site, while another child site request a complete abort. The way in which the abort conflicts are resolved depends on the state in which a subprocess resides at the time an abort request is received. Figure 3 shows the state diagram of a subprocess. A subprocess can reside in the following states:

- Executing, i.e. the subprocess is being executed normally by the WFMS,
- On Hold, i.e. the subprocess is stopped while a compensating process is being created by the transaction system. This state is not shown in the diagram to keep it readable as the subprocess is always put on hold whenever an abort request is issued and a compensating global transaction is being calculated.
- Compensating partially, i.e. the WFMS is executing a compensation process that has been created by the transaction system based on a partial abort request from the failing subprocess,
- Compensating completely, i.e. the WFMS is executing a compensation process that has been created by the transaction system based on a complete abort request from the failing subprocess, and
- Finished, i.e. the subprocess has been completely executed.

A backward or forward abort request can be received while the subprocess resides in any of the five states. The reaction to the backward or forward abort request can be seen in the state diagram: If a subprocess is in execution or finished state, no abort conflict can occur and an abort request is handled as described in the previous
subsection. However, abort conflicts can occur when the subprocess is in one of the other three states and are handled in the following way, see also Figure 3:

- If the subprocess is in the on hold state and thus a compensating global transaction is being created, the following table shows the occurring conflicts and the reaction of the distributed global transaction support:

<table>
<thead>
<tr>
<th>Creation of compensation in partial abort mode</th>
<th>Receiving abort request in partial mode</th>
<th>Receiving abort request in complete mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Finish creation and execution of compensation, then handle the received abort request</td>
<td>Cancel creation, then create and execute the compensation in complete mode</td>
</tr>
<tr>
<td>Creation of compensation in complete abort mode</td>
<td>Ignore receiving abort request</td>
<td>Ignore receiving abort request</td>
</tr>
</tbody>
</table>

- If the subprocess is executing a compensating global transaction, i.e. in the compensating partially or compensating completely states, the received abort request is “remembered” and handled after the compensation process has been executed, i.e. concurrent aborts will be handled one after the other.

An example of two concurrent abort requests is shown in Figure 4 in which the failing action on site 4 requests a partial abort and the failing action on site 5 requests a complete abort.

Both failing sites will issue a backward abort request to site 3. Suppose the partial backward abort request is handled first. During the creation of the partial compensation request, a forward abort is issued to site 4. Site 4 has already handled a complete abort request so no additional abort is necessary at site 4. When the backward complete abort request from site 4 is received by site 3, which is executing the partial compensation request, it first finishes the partial compensation (leaving only the savepoint as the rest of the actions are compensated) after which it handles the complete abort request. In this case no forward abort request is issued as site 5 has already been compensated.

The architecture that has been designed to support the discussed distributed transaction model is described in the next section.
5 Architecture of the Distributed Global Transaction Support

The architecture of the distributed global transaction (DGTS) support does not differ much from the non-distributed global transaction support (GTS). As described before, the necessary communication protocol needs to be added. A non-distributed GTS consists of two parts: a global transaction manager (GTM) and global transaction (GT) objects [12]. Each global transaction, i.e. workflow execution, is represented by one GT object. GT objects are dynamic objects that are created when a global transaction is started and destroyed when a global transaction finishes. They keep track of workflow executions by administrating events like starting and ending of global transaction steps.

5.1 Distributed Global Transaction Manager

The distributed global transaction manager (DGTM) is composed of multiple modules. The most important module with respect to the transactional support for distributed workflow executions is the communication module. It is responsible for the coordination of the different sites in case of an abort that involves multiple sites. The WFMSs involved in a distributed workflow execution have a communication mechanism that allows them to interoperate, e.g. using interface 4 specification of the Workflow Management Coalition [22]. This communication mechanism could be used by the DGTS to signal forward and backward aborts to other sites, but that requires the WFMS to handle some of the transactional functionality, reducing the orthogonality between the DGTS and the WFMS. The architectural choice has therefore been made to have a separate and direct communication mechanism between GTS systems. This makes the DGTS and the WFMS as orthogonal as possible, thereby shielding the WFMS from transactional issues and increasing the portability of the transactional support system. The complete architecture with the communication infrastructure is shown in Figure 5a. In the figure, the solid arrows represent the communication between the WFMSs and the dotted arrows represent the communication between the different GTS systems.

The logic module of the DGTM contains the algorithms that are used to calculate compensating global transactions for one site. A formal specification of those algorithms, i.e. for non-distributed workflow executions, can be found in [13].

5.2 Extended Architectural Considerations

The workload of the DGTS depends on the amount of failures that occur in workflow executions. If few failures occur, it is not necessary to have a transaction system on every site where a WFMS is running which only wastes resources. In case of many failures, it is more efficient if the WFMS can use more than one transaction support system. Both requirements can be satisfied by making it possible for any WFMS to use any GTS system that is available, i.e., one WFMS may use multiple GTS systems and one GTS system can be used by multiple WFMSs. This can be realized using a middleware system such as a CORBA [19] compliant object request broker. In this case it is possible to arbitrarily distribute the GTS
systems over multiple sites. The GTS systems can then be called by any of the WFMSs, which may or may not be on the same site. This way a very flexible and adaptable architecture is offered, of which an example is shown in Figure 5b. Whenever more or less GTS systems are required, a new GTS system can be instantiated or an existing one can be shut down. For the non-distributed global transaction support system the arbitrary distribution of GTS systems is already realized in the WIDE project.

6 Conclusions and Future Work

This paper describes a model and architecture to provide transactional support for distributed workflow executions. The transaction model relies on compensating actions to semantically undo the effects of completed actions. Specifying savepoints in the workflow process allows to partially compensate a workflow. The WIDE global transaction support system is extended with a communication protocol to provide support for distributing workflow executions over multiple sites. The communication protocol coordinates the calculation of the sites that are involved in an abort, in which abort mode the compensation for those sites must be performed, and the enactment of the distributed global transaction support system at those sites.

In the WIDE project, the non-distributed version of the global transaction support system has been implemented in a prototype environment to support the commercial workflow management system FORO, marketed by SEMA Group [10]. The environment is based on ILU [16], which is a CORBA compliant object request broker and the Oracle database management system [20]. The realization of the extensions to support the distributed execution of workflows as described in this paper is currently considered and can be achieved with little effort. Because of the modularity of the global transaction support system and the orthogonality to the workflow management system, only the communication module needs to be specified and implemented, which can be easily integrated into the global transaction support system.
The transaction model and architecture are orthogonal to the workflow management system and can therefore be easily applied to other application areas dealing with long-running processes. The work presented in this paper is currently under consideration to be used in the CrossFlow project [7].

References