A CONCURRENCY CONTROL MODEL FOR MULTIPARTY BUSINESS PROCESSES

Juha Puustjärvi
Lappeenranta University of Technology, Skinnarilankatu 34, Lappeenranta, Finland

Keywords: Web services, WS-Coordination, concurrency control, workflows, business processes, advanced transaction models.

Abstract: Although the issue of atomicity of multiparty business processes is well understood and widely studied, the concurrency control issues of multiparty business processes is not studied nor well understood. In this paper, we restrict ourselves on this issue. First we motivate the need of concurrency control in this context. Then, we present a liberal correctness criterion, called set-serializability and a scheduler based on timestamp ordering rule that produces set-serializable executions. Technically the scheduler is very simple and it can be easily integrated with the protocol that ensures the atomicity of the multiparty business processes. In implementing the atomicity protocol and the scheduler we utilize the WS-Coordination, which is a general and extensible framework for defining protocols for coordinating activities that are part of business processes.

1 INTRODUCTION

Web services are self-describing modular applications that can be published, located and invoked across the Web (Newcomer, 2002). Once a service is deployed, other applications can invoke the deployed service. The service can be anything from a simple request to complicated business process.

Another nice feature of web services is that new and more complex web services can be composed of other web services (Daconta et al, 2003; Marinescu, 2002). However, in many cases composed web services are useful only if they can be processed atomically.

WS-Coordination (Singh & Huns, 2005) is a general and extensible framework for defining protocols for coordinating activities that are part of business processes. In particular, *WS-BusinessActivity* (Singh & Huns, 2005) is a protocol that exploits WS-Coordination to define coordination type for long-duration business transactions.

The long duration of business activities prohibits locking data resources to make actions hidden from other concurrent activities, and so the transactions supported by WS-BusinessActivity do not have isolation characteristics. The atomicity of the transactions supported by WS-BusinessActivity is based on compensating transactions (Garcia-Molina, 1983).

Although the issue of atomicity of composed Web services and multiparty business processes are widely studied, (e.g., (XLANG, 2001; XAML, 2003; BTP, 2002; WSFL, 2003; BPEL, 2004)) the issue of isolation in this context is not addressed. We therefore focus on analysing concurrency control issues of multiparty business processes.

In our analysis, similar to (Puustjärvi, 2001), we view multiparty business processes from workflows point of view, i.e., we view workflows as collections of tasks that are organized to accomplish some business process. As a result we can easily map workflows into structured transactions: the transaction represents the workflow and its subtransactions represent the tasks of the workflow. In particular the goal of this paper is to:

1. to demonstrate the need of concurrency control in the context of multiparty workflows,
2. to develop an appropriate correctness criterion for the execution of concurrent multiparty workflows,
3. to develop an appropriate concurrency control method for managing multiparty workflows, and

Puustjärvi J. (2008).
A CONCURRENCY CONTROL MODEL FOR MULTIPARTY BUSINESS PROCESSES.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 209-215
DOI: 10.5220/0001513102090215

4. to demonstrate how WS-Coordination can be utilized in implementing a scheduler for multiparty workflows.

On the other hand, our analysis will show that the concurrency control of multiparty workflows is a trade off between

- workflow execution correctness,
- workflow system performance, and
- the simplicity of workflow specification and management.

Our viewpoints are presented in the following sections as follows: First, in Section 2, we introduce a multiparty workflow and illustrate its isolation requirements. Then, in Section 3, we give a short introduction to concurrency control methods and schedulers. In Section 4, we focus on concurrency control correctness criteria. In Section 5, we first introduce our developed concurrency control criterion, called set-serializability, and then we describe the concurrency control method that supports set-serializability. In addition we demonstrate how this method can be integrated with an atomicity protocol and how WS-Coordination can be utilized in developing the runtime environment for multiparty workflows. Finally, Section 6 concludes the paper by discussing the advantages and disadvantages of our developed solutions.

2 MOTIVATION

We now represent a multiparty business process which correct execution requires coordination to ensure its atomicity and as well as its isolation.

Consider an oil broker on the Web. In order for the broker to deliver oil, the broker requires additional value-added services provided by third parties, such as chemical provider, shipping, payment financing, and casualty insurance. The broker will not agree to the delivery of oil until all of these services are available, i.e., the correctness requires the execution to be atomic.

From technology point of view the software providing the multiparty business process needs to coordinate with each of the participating Web services. These include (1) the chemical provider's inventory system; (2) credit institution to check customer creditability; (3) an insurance policy service to insure the product being shipped; (4) a financing service to ensure payment; and (5) a transportation service to guarantee timely shipment and delivery.

We now describe this multiparty business process by a workflow in Figure 1. (By a workflow instance we refer to an execution of a workflow) Its task Enter order provides an interface for customers. It records orders, which include (among other things) information of the ordered chemicals and the deadline for the delivery. Then two parallel tasks are processed: Purchase chemical task updates the inventory of the chemical provider and Check creditability task checks the customer's credit information from a credit institution. After their successful processing, Order transportation task orders the delivery from a transportation company, and finally transportation is insured and the customer is charged.

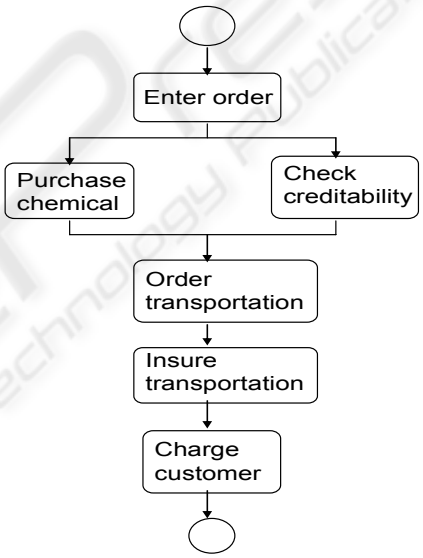

Figure 1: Oil Broker's business process.

To illustrate workflow isolation requirements, we now give three isolation requirements for this oil broker's workflow and consider the effects of their violations.

Case 1: Businesses often enumerate events with unique sequence numbers, and so there may be a reguirement that those numbers (given in the task Enter order) must constitute a monotone series with no gaps. So, a new order number cannot be issued until it is sure that the previous workflow will not fail. However, as the workflow may fail at any time during its execution, the only way to ensure that there are no gaps is to execute the workflows serially.

Case 2: Assume that the balance of unpaid bills of a customer has a predefined upper limit. The balance is checked in the task Check creditability, and if the new order would cause an overdraft then the work-flow is aborted (a semantic failure). Otherwise, the new balance is updated in the task Charge customer. Now, to ensure that the limit will not be executed as a result of two or more concurrent workflows of the same customer, the workflows pertaining to the same customer should be processed in a serializable way, or at least the tasks Check creditability, Insure transportation and Charge customer pertaining to the same client should be processed in a serializable way.

Case 3: Assume that the existence of an ordered product is checked in the task Purchase chemical, and that the ordered product is not removed from chemical provider's inventory database until in the task Order transportation task. Now, to ensure that the ordered product is still in the inventory, the task Purchase chemical and Order transportation should be executed as one transaction, or at least they should constitute a unit of isolation.

3 TRADITIONAL CONCURRENCY CONTROL METHODS

We now give a short introduction to the notions that are related to concurrency control. In particular, we consider the notions that we use in later analysis.

Concurrency control is the activity of coordinating the actions of processes that operate in parallel, access shared resources, and therefore potentially interfere with each other (Bernstein et al, 87). *A scheduler* is a program that controls the execution of concurrent activities (Gray & Reuter, 93). When it receives an operation it may immediately schedule it, delay it, or reject it. Each scheduler usually favours one or two of these choices.

Almost all types of schedulers have an aggressive and a conservative version. *An aggressive scheduler* tends to avoid delaying operations, and so it loses the opportunity to reorder operations it receives later on. *A conservative scheduler* tends to delay operations, and so it can reorder operations it receives later on.

Locking is the most common type of schedulers (Bernstein & Newcomer, 1997). The idea behind locking schedules is intuitive: each resource to be accessed (e.g., data item or a web service) has a lock associated with it, and before an activity (e.g., transaction or workflow) may access a resource, the scheduler first examines the associated lock. If no activity holds the lock then the scheduler obtains the lock on behalf of the re-questing activity.

With timestamp methods a unique time stamp is assigned to each transaction. Transactions are then processed so that their execution is equivalent to a serial execution in timestamp order. This concurrency control mechanism allows a transaction to access a data item only if it had been last accessed by an older transaction; otherwise it rejects the operation and restarts the transaction.

Each type of scheduler works well for certain types of applications. We will show that an aggressive scheduler based on timestamp ordering method will work well with multiparty business processes. However, the traditional correctness criterion would be overly restrictive and therefore we will introduce a more liberal correctness criterion.

4 CONCURRENCY CONTROL CRITERIA

A natural and sufficient criterion for isolation correctness is that the execution is serializable, i.e., equivalent to a serial execution. Moreover this traditional criterion is intuitive and clear. However, though it is suitable for traditional transactions it would overly restrict the concurrency of long lasting activities such as multiparty workflows. However, by using semantic information it is possible to weaken the serializability criterion, and yet ensure execution correctness. On the other hand, analogous with traditional semantic concurrency control models (Lynch, 1983; Garcia-Molina 1983) the use of semantic information makes the specification as well as the management of the system more complex.

With multiparty workflows the requirements for concurrency control significantly deviates from those used with databases. In particularly there are no consistency constraints between the data stored in communicating applications but rather (as illustrated in Section 2) the workflows may interfere with each others through accessing dirty data (i.e., data that is written by uncommitted activities). Therefore neither the correctness criterion nor the concurrency control methods (e.g., two-phase locking) developed

for databases are suitable for managing multiparty workflows.

We next illustrate how we can capture semantic information from multiparty workflows and use it in developing an appropriate correctness criterion and concurrency control method for multiparty business processes.

5 THE MODEL

In this section we introduce our model which describes our developed correctness criterion and concurrency control method. In addition we describe how it can be implemented by extending the 2PC-protocol (Bernstein and Hadzilacos, 1987) that is used for ensuring the semantic atomicity of multiparty business processes.

5.1 Serializability-sets

As the analysis of the oil broker's workflow in Section 2 showed, there is no need for requiring global serializability of workflows. By globally serializable execution we refer to the execution, which is equivalent to a serial execution of workflows. Instead, it seems that a sufficient isolation requirement is that certain sets (comprised of tasks or workflow instances) should be executed in a serializable way. Each such a set we call a *serializability set*.

In order to illustrate the forms of serializability sets we now assume that we have two workflows, denoted by *Wi* and *Wj*. For example, the workflow Wi could be the oil broker's workflow presented in Section 2.

The tasks of the workflow *Wi* are denoted by *Ti,1, ... Ti,m,* and anagolously the tasks of workflow *Wj* are denoted by *Tj,1, ... Tj,n.* So workflow *Wi* is comprised of *m* tasks and the workflow *Wj* is comprised of *n* tasks.

We next characterize the nature of serializability sets by making the difference between four kinds of serializability sets:

1. One or more tasks of the same workflow, say *Ti,s* and *T i,k*, have to be executed serially (e.g., cases 2 and 3 of Section 2). So the serializability set is of the form *{Ti,s, Ti,k,}*.
2. The instances of one workflow, say the instances of workflow Wi have to be executed serially (e.g., case 1 of Section 2). So the serializability set is of the form {Wij}.
3. Two or more task instances, say T i,s and Ti,k, from different workflows have to be executed serially. So the serializability set is of the form {Ti,s and Ti,k}.
4. The instances of two or more workflows, say the instances of workflows *Wi* and *Wk*, have to be executed serially. So the form of the serializability set is of the form *{Wi, Wj}*.

We denote by S the set of consisting of all serializability sets, i.e., if there are n serializability sets denoted by S1, …,Sn, then S={S1, …,Sn}. It clear that each tasks belongs into zero, one or more serializability sets.

5.2 Set-serializability Criterion

Now we can specify our used isolation correctness criterion:

Set-serializability Criterion. The execution of a set of workflows is *set-serializable*, if the execution of the workflow and tasks instances in each serializability set is serializable.

Note that this criterion is much more liberal (i.e., allows much more concurrency), than the traditional serializability criterion. The reason is that only those workflow instances or task instances, which really need serializable execution are enforced to be serializable. In contrast with traditional serializability criterion it is assumed that all activities require serializable execution. Hence, the traditional serializability criterion is called syntactic concurrency control model, which means that no semantic information of transactions are given. Instead, our model is based on a semantic concurrency control model, in which it is assumed that the workflow designer gives semantic information through serializability sets. In practice this means that a workflow designer has to know the workflow well enough to be able to specify appropriate serializability sets.

5.3 Enforcing Set-serializable Executions

In order to identify different instances of the same workflow, an instance identifier is attached to each workflow instance. In our solution the identifier is determined according to the workflow identifier and the time the workflow instance starts. Similarly, tasks instances are identified by workflow identifier, task identifier and timestamp.

We next consider how we can use timestamp method in scheduling the workflow instances. The sufficient requirement is that the task instances of each serializability-set are executed in the order determined by their timestamps.

As each task corresponds to the request of a web service, the service provider has to serve the request in the order determined their time-stamps. This means that there must be a standard module (i.e., a scheduler) that can be combined to the web service. In other words, assume that a task Ti belongs to one or more isolation cluster. Then there must be a scheduler, say scheduler S(ti), which follows the following rule in handling requests:

Request Scheduling Rule: accept the execution request of the task Ti belonging to serializability sets Si, …,Sn, if the timestamps of the last accepted request of the serializability sets S1, …, Sn are older than the timestamp of Ti, and otherwise reject the request.

If a request (a task) is rejected then the corresponding workflow instance has to be aborted and restarted. The abortion requires that the tasks of the workflow instance which are already processed have to be compensated. This, however, requires no extra modules because the mechanisms which support the atomicity of the workflows support abortions through compensating actions.

In Figure 2 we illustrate the scheduler of task Ti. It is assumed that task Ti is processed by Web service Wi, and therefore scheduler Si locates on the Web service Wi. The figure illustrate the case where task Ti belongs to serializability set S1, S2 and S3, and therefore the data structure maintained by the scheduler Si includes the timestamps (denoted lastS1, lastS2 and lastS3) of the last accepted requests of the serializability sets S1, S2 and S3.

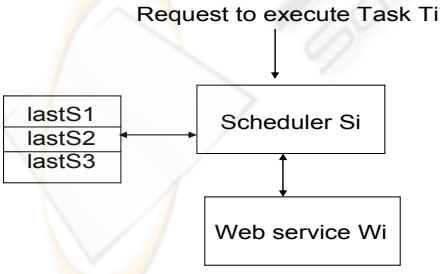

Figure 2: A scheduler attached to Web service.

5.4 Exploiting WS-Coordination in Enforcing Set-serializability

A way to coordinate the activities of Web services is to provide a Web service which function is to do the coordination. In order to alleviate the development of such coordinators WS-Coordination provides a specification that can be utilized in developing the coordinator. According to the WS-Coordination a coordinator is an aggregation of the following services:

- An activation service: defines the operation that allows the required context to be created. In particular, a context identifier is created and passed to the services that participate to the same coordination.
- A registration service: defines the operation that allows a web service to register its participation in a coordination protocol.
- A coordination protocol service for each supported coordination type.

In Figure 3, the architecture (following the specification of WS-Coordination) of the coordinator that supports multiparty workflows is presented.

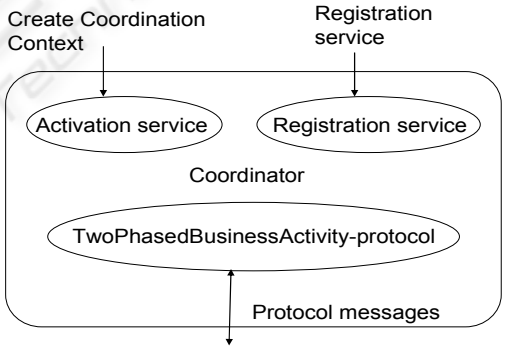

Figure 3: The components of the coordinator.

After an application has created a coordination context, it can send it to another application. The context contains the information required for the receiving application to register into the coordination. In principle an application can choose either the registration service of the original application or use some other (own) coordinator. In the latter case the application forwards the context to the chosen coordinator.

In our solution each participating application uses its own coordinator. We illustrate this by the example presented in Section 2. For simplicity, in

the Figure 4, we only present oil broker's communication with two participants (chemical manufacturer and credit institution).

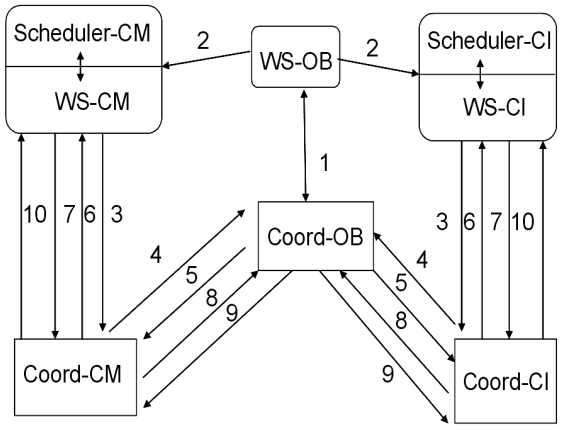

Figure 4: The coordination structure between schedulers, Web services and their coordinators.

In the figure WS-OB stands for oil broker's Web service, WS-CM stands for chemical manufacturer's Web service, WS-CI stands for credit institution's Web service, Coord-OB stands for oil broker's coordinator, Coord-CM stands for chemical manufacturer's coordinator and Coord-CI stands for credit institution's coordinator.
The communication proceeds as follows:

1. Oil broker's Web service asks its coordinator to create a coordination context for Atomic Transaction -type coordination (2PC-type coordination), and then the coordinator returns the context which includes information where its registration service can be found. In addition, the context includes a timestamp on which the scheduling will be based on.
2. Oil broker's Web service sends oil purchase message to manufacturer's Web service and creditability request to credit institution's Web service. Both messages include context information. As the task Purchase chemical and Check creditability belongs to the same serializability set, the Web services have to schedule them., i.e., ensure that they are executed in the order determined by their timestamps. If there is a violation in the timestamp order then the compensation protocol is started which undoes the effects of the tasks that are already executed (note that in this example there are no such tasks).
3. Oil broker's Web service, chemical manufacturer's and credit institution's Web services send the context information to their own coordinators.
4. Manufacturer's coordinator and credit institution's coordinator register to oil broker's coordinator.
5. Oil broker's coordinator sends the request message to chemical manufacturer's and credit institution's coordinator.
6. Chemical manufacturer's coordinator and credit institution's coordinator request their Web services to execute the activity.
7. Chemical manufacturer's Web service and credit institution's Web service inform their coordinators whether the execution failed or not.
8. Chemical manufacturer's coordinator and credit institution's coordinator informs oil broker's coordinator whether the execution failed or whether it was successfully executed.
9. If there were no failure then oil broker's coordinator sends the Commit-message to chemical manufacturer's Web service and credit institution's Web service; otherwise it sends the Abort-message.
10. Chemical manufacturer's coordinator informs its Web service and credit institution's coordinator informs its Web service whether the multiparty workflow is committed or aborted. In the case of abortion the web services execute the compensating transactions which undo the effects of the executed transactions.

6 CONCLUSIONS

With traditional database transactions the need for concurrency control is usually motivated by the phenomena such as lost update and inconsistent retrieval, and the commonly used correctness criterion for schedules (histories) is serializability. With multiparty business processes the requirements for concurrency control significantly deviates from those used with databases. Therefore neither the correctness criteria nor the concurrency control methods developed for databases are suitable for multiparty business processes.

A nice feature of our developed concurrency control model is that it enforces scheduling only on those workflow instances that really need it. In contrast with traditional database systems all the transactions are enforced under scheduling which significantly decreases the throughput of the system.

Another nice feature of our concurrency control method is that it can be easily integrated with the atomicity protocol. In addition, technically the timestamp ordering scheduler is very simple (e.g., compared with the locking schedulers in database systems) as it only checks whether request are served in the order determined by the timestamps.

A disadvantage of our semantic concurrency control model is that the business process designer is burden with defining the serializability-sets. However, obviously there is no way around of burden the designer if some semantic concurrency control model is used.

REFERENCES

Bernstein, P. V. Hadzilacos, V. & N. Goodman, N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.

Bernstein, P. & Newcomer, E. Principles of Transaction Processing. Morgan Kaufmann Publisher. 1997

BPEL, 2004. BPEL4WS – Business Process Language for Web Sevices. http://www.w.ibm.com/developersworks/webservices/library/ws-bpel/BPEL, 2004.

BPEL, 2004. BPEL4WS – Business Process Execution Language for Web Sevices. http://www.w.ibm.com/developersworks/webservices/library/ws-bpel/

BTP, 2002. BTP- Business Transaction Protocol, http://www.oasis-open.org/committees/business-transactions/documents/primer/.

Daconta, M., Obrst, L. & K. Smith, K. The semanticweb. Indianapolis: John Wiley & Sons. 2003

Garcia-Molina, H. Using semantic knowledge for transaction processing in a distributed database. ACM Transactions on Database Systems, 8(2):186-313, 1983.

Gray, J. & Reuter A. 1993. Trasaction Processing: Concepts and Techniques. Morgan Kaufman.

Lynch N.. Multilevel atomicity – a new correctness criterion for database concurrency control. ACM Transactions on Database Systems, 8(4):65-76.

Marinescu, D. Internet-based workflow anagement. John Wiley & Sons, 2002.

Newcomer E., 2002. Understanding Web Services Addison-Wesley.

Puustjärvi, J. Workflow concurrency control. The Computer Journal, 44(1), 2001.

Singh, M & Huhns, M. Service Oriented Computing: Semantics, Processes, Agents. John Wiley &Sons, Ltd. 2005.

WSFL, 2003. WSFL- Web Services Flow Language. http://www.ebpml.org/wsfl.htm

XAML, 2003. Transaction Author Markup Language (XAML). http://xml.coverpages.org/xaml.html

XLANG, 2001. XLANG–Web Services for Business Process Design. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm