Control and Cybernetics

vol. **40** (2011) No. 3

Transaction mechanisms in complex business processes[*][†]

by

Krzysztof Jankiewicz, Kamil Kujawiński, Mateusz Mor and Tadeusz Morzy

Poznań University of Technology, Institute of Computing Science
Pl. Marii Skłodowskiej-Curie 5, Poznań, Poland
e-mail: {Krzysztof.Jankiewicz, Tadeusz.Morzy}@cs.put.poznan.pl

Abstract: The importance of systems based on the SOA architecture continues to grow. At the same time, in spite of the existence of many specifications which allow for the coordination of business processes functioning in the SOA environment, there is still lack of solutions that allow the use of system mechanisms of transaction processing. Such solutions should entail the simplification of construction of business processes without affecting their capabilities. This article presents a Transaction Coordinator environment, designed and implemented as a response to these needs.

Keywords: SOA, transactions, business processes, concurrency control, atomicity, compensation.

1. Introduction

The introduction of transaction mechanisms to the specification and the execution of complex business processes in the SOA environment would allow users, similarly as in the case of database systems and distributed information systems, to define computations with the use of semantic commands that control transactions. In systems based on SOA, we deal with asynchronous, complex and lengthy business processes, where participants (services) are loosely related. Therefore, in these systems the classic model of flat transactions and nested transactions, for which the entire transaction is a unit of atomicity, consistency, isolation, and durability is inadequate and inappropriate. An important problem is the irreversibility of the execution of certain operations, which is similar to the irreversibility of the real actions in the databases. For this reason, compensation mechanisms are needed for the irreversible effects of the transaction or its fragments. Another problem, in the case of business processes in the

[*]Submitted: March 2011; Accepted: August 2011.

[†]The research presented in this article was partially supported by the European Union in the scope of the European Regional Development Fund (ERDF) program no. POIG.01.03.01-00-008/08.

SOA environment, is the need to define complex, nested structures built out of many dependent or independent parts, which have various characteristics, for example concerning the impact of their execution on the final effect of the whole business process. The lack of mechanisms allowing for the specification and system support of the above mentioned structures, makes the definition of complex business processes complicated, and the mistakes made in this phase expose business processes to faulty execution.

This article comprises a description of the implemented software environment, named Transaction Coordinator, which allows for the specification and execution of complex business processes, in which particular fragments can be executed considering declared properties of transaction processing.

The structure of the article is as follows: Section 2 includes the characteristics of business processes, in Section 3 existing specifications, which are related to transaction processing in the SOA environment, and results of their analyses are shown, Section 4 discusses the architecture and modules of the Transaction Coordinator, results of conducted tests are presented in Section 5, summary and directions of future works are given in Section 6.

2. Characteristics of business processes

2.1. The characteristics of business processes in the SOA environments

While in database management systems the processing model dominates based on transactions with ACID properties (see Bernstein, 1986), in the SOA environments the execution of business processes requires more flexible solutions (see McGovern, 2003). This results from the fact that SOA environments are based on Web services, which in their nature are loosely related, the duration of processes can be measured in days or months, and they have a complex definition built on many nested structures with various characteristics. The next significant difference is the handling of failures. In the case of database management systems handling of failures, which takes place within the transaction, usually consists of a series of system operations aimed to return to the state before the transaction. In business processes actions taken because of a failure may have a different character. In many cases it is not possible to undertake system operations – this is because the form of these actions is strongly related to the place of the failure in the business process. Let us consider the example of buying goods in an Internet shop; this type of process consists of many stages: making the order, paying for the order, producing the goods, transporting the goods to the warehouse, delivering the goods to the customer. From a business point of view totally different actions should be taken when the process fails due to the destruction of goods during the transport to the warehouse (the goods should be reproduced and transported to the warehouse once again and the process can be continued) and when the "failure" is caused by the resignation of the customer when receiving the goods (goods may be placed back in warehouse). This example shows that on some level of abstraction the system approach based on transactional mechanisms is inappropriate. This does not change the fact that mechanisms known for transactions are needed. Let us imagine a complexity of business process, in which after each step, every invoked service, there is a procedure of exception handling, which can appear at this point. This type of process, from the definition point of view, would be very complicated. However, most business processes are made up of numerous complex tasks (sub-processes), based, for example, on invoking a series of Web services. The extraction of these tasks in the form of transactions, which have particular properties and in a system manner react to failures, which take place within them, would allow to simplify the definition of business processes and allow their creators to concentrate on business issues.

2.2. The transaction model in the SOA environments

Transactions occurring in business processes executed in the SOA environment have to meet requirements related to the characteristics of business processes.

2.2.1. The complexity of the business process

We assume that the business process may be constructed with many transactions. The effects of each transaction are instantly accessible for other business processes and in this context the model of the business process corresponds with sagas model (see Garcia-Molina, 1987). A failure of the business process requires from the creator of the process to define appropriate actions, which will take into account the fact that until the failure some of the transactions have been done and committed. These actions are defined by the creator of the process (in contrast to system actions), because usually they have a business character. On the other hand, the creator of the business process does not have to worry about the actions executed within transactions, which have not been finished until the failure. Their roll back or compensation is done by the transactional system mechanisms.

2.2.2. Optional and mandatory components

Every transaction, similarly as a business process, can be constructed with child transactions. Each child transaction has a property which informs whether its correct completion is a requirement for the success of the parent transaction. This allows for a simple definition of mandatory and optional components of the transaction. Let us consider the parent transaction which has three, sequentially executed, child transactions responsible for: booking a hotel room, purchasing a plane ticket and organising taxi to the airport. Because we do not want the failure of the child transaction connected with organising a taxi to be the reason for the failure of the whole parent transaction, we can define it as optional.

2.2.3. Dependence and independence of components

The next property of child transactions is their dependence or independence of the final success of the parent transaction. In the case of independent child transactions, the effects of their finalised actions, do not depend on the success or failure of the parent transaction. In the case of dependent child transactions, actions undertaken by them may be rolled back or compensated not only on the basis of withdrawal of the child transaction, but also as a result of failure of the parent transaction. Let us once again consider the parent transaction from the previous paragraph. Let us assume that booking a hotel room and purchasing of a plane tickets requires a non-returnable payment for the appropriate service. Therefore, the child transaction, responsible for the fulfillment of the mentioned payment, positioned before the other child transactions will be independent – it will not be an object of withdrawal or compensation, even in the case of a failure of the parent transaction. The remaining child transactions will be dependent. For example a finished child transaction involving booking a hotel room will be compensated in the case of a failure of the child transaction, responsible for the purchase of a plane ticket, which, as mandatory, will cause the withdrawal of the parent transaction.

2.2.4. Partial rolling back of a transaction

To increase throughput of the system which executes business processes, every transaction can define savepoints. These allow, in the case of failure, for a partial rollback (compensation) of completed actions, and then the resuming of actions, through re-execution of the same actions or the alternative ones. For example, failure to purchase a plane ticket within the discussed transactions, can initiate the execution of alternative actions, purchasing a ticket through other airlines and does not determine an automatic withdrawal of the whole parent transaction.

3. Existing proposals and solutions

3.1. The current specifications and standards

The problem of transaction processing in systems based on SOA has already been noticed a few years ago. As a result, a number of protocols and specifications have been developed, supporting the implementation of transaction processing. Among them are those developed by the OASIS WS-TX Technical Committee: *WS-Coordination* version 1.2 (*WS-Coor*) (see Feingold, 2009), *WS-AtomicTransaction* version 1.2 (*WS-AT*) (see Little, 2009) and *WS-BusinessActivity* version 1.2 (*WS-BA*) (see Freund, 2009). In addition, the issues related to the processing of long-term business processes, which take into account the need for compensation, have been addressed in the *Web Services Business Process Execution Language* (*WS-BPEL*) (see Alves, 2009).

3.2. Results of analysis

On the basis of analysis of current solutions and specifications we can notice that each of them has some limitations and must meet a number of conditions.

WS-AtomicTransaction (*WS-AT*) is a specification that is intended only for short-lived transactions and is not suitable to coordinate a long-running business activity. This specification allows for the coordination of services, which must comply with many features. For example, actions taken by these services prior to commitment must be tentative, typically they are neither persistent nor made visible outside the transaction. Only a commitment of a transaction directs participants to make the tentative actions final; then, these actions may be made persistent and visible outside the transaction. This specific way of functioning of participants within a transaction is a serious limitation that cannot be met in many cases, significantly limiting the possibility of using this specification.

WS-BusinessActivity (*WS-BA*) is a specification designed for long-running distributed activities. Similarly to *WS-AT*, *WS-BA* is also based on *WS-Coor*, therefore in both cases control of the processes is conducted by a coordinator, and the communication between the particular participants of the activity and the coordinator is done with defined protocols. Nevertheless, in contrast to the *WS-AT* specification, which defines the protocols used in cooperation between the coordinator and the process initiator, and also between the coordinator and other participants (Web services), the *WS-BA* specification defines only the protocols used for communication between the coordinator and participants of an activity other than their initiator. This is due to the fact that rules of aborting or committing business transactions can be very complex and, in accordance with the intention of the *WS-BA* specification, they belong to the sphere of business. This makes implementation of *WS-BA* generally very complicated. The creator of a business process must develop his own protocol of cooperation between the initiator and the coordinator of an activity, as well as appropriate protocol services for both sides. An alternative solution is total integration of the coordinator and the initiator of the activity. This, however, leads to mixing of elements from business (business process definition) and system mechanisms (coordination of the transaction), which significantly affects the simplicity and clarity of the process definition. Another difference between *WS-AT* and *WS-BA* is that the latter does not impose restrictions on participant functioning – actions performed by participants may be immediately persistent and visible outside the transaction, regardless of how a business process ends. As a result, a failure of a business process requires appropriate compensation. According to the *WS-BA*, the participants of an activity are responsible for the compensating operations. The initiation of a compensating operation is triggered by a message sent to participants. The compensation rules of *WS-BA* only allow for full compensation to be undertaken by all participants or selected ones. There is no possibility to compensate specific actions.

WS-BPEL is a specification of declarative markup language, used to describe, coordinate and implement complex business processes using Web services. The *WS-BPEL* language allows to define a long-term activity, called *Long-Running Transaction* (*LRT*). Unfortunately, the solutions of *WS-BPEL* assume that elements, such as exception handling or invoking compensation actions, must be prepared by the creator of the business process. Thus, like in *WS-BA* specifications, we face a situation in which business components are mixed with system actions. Moreover, in contrast to the *WS-BA* specification, for which compensation is triggered with one simple message, the creator of a business process expressed in *WS-BPEL* must invoke appropriate compensation services. As a result, the definition of properly constructed business processes is a complex task.

Moreover, it should be noted that in no way does *WS-BPEL*, as well as *WS-BA*, relate to the problems of concurrent execution of multiple business processes.

The analysis of the specifications listed above leads to the conclusion that it is appropriate to propose solutions that will significantly support creation of business processes, separating the business layer from the system actions. This solution should allow for a proper coordination of business processes at the stage of acceptance (or rejection), it should provide an opportunity to make a partial compensation and it should take into account the need for the coordination of concurrently executed processes. To achieve these goals, we have designed and implemented the Transaction Coordinator. It allows for:

- application of the control structures within the business process code associated with the concept of a transaction,
- definition of a transaction operating within SOA environment,
- management of transactions taking into account their declared properties,
- control of concurrent execution of transactions,
- management of a transaction completion.

It should be noted that the problem of transaction management in the SOA environment has become the subject of analysis of many studies. We can mention the frequently quoted article by Lieberman (2006), chapters in books concerning SOA architecture (see McGovern, 2003, 2006; Krafzig, 2004), or conference presentations (see Alrifai, 2006; Choi, 2005; Younas 2006). In these studies the specification of business processes in service-oriented environments is analysed, coordination concepts and prototypes of services coordinating transaction processing are proposed. Some ideas are very interesting. However, in most studies only isolated problems of transaction properties are considered. For example, issues related to the atomicity of transactions or the usage of compensation are analysed, but omitted are issues related to the concurrent realisation of business processes. Furthermore, some authors analyse relatively simple transaction models, which in our opinion are inadequate for defining complex business processes.

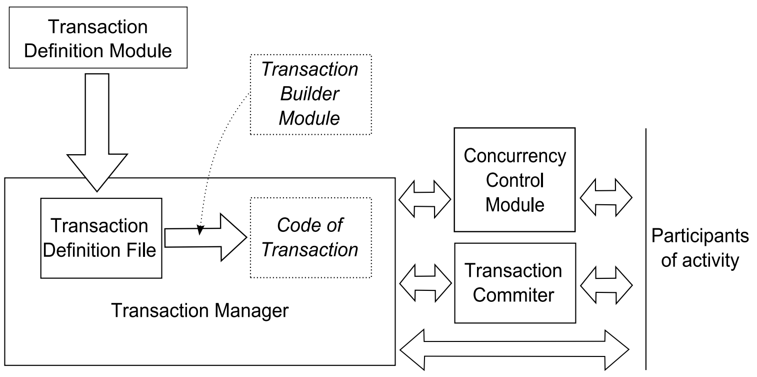
Transaction Definition Module
Transaction Builder Module
Concurrency Control Module
Transaction Definition File
Code of Transaction
Participants of activity
Transaction Commiter
Transaction Manager

Figure 1. The Transaction Coordinator architecture

4. The Transaction Coordinator

The Transaction Coordinator has a modular design presented in Fig. 1. The Transaction Definition Module helps the end-user to define a business process using the control structures of transaction processing. The output from the module is the Transaction Definition File, which is the definition of the BPEL process, supplemented by transaction processing commands, and other statements related to the business process definition. The task of the Transaction Manager is the execution of the Transaction Definition File. In particular, the role of the Transaction Manager can be played by any BPEL server allowing for interpretation of transaction processing commands. The Transaction Builder Module operates under the Transaction Manager and performs proper interpretations of the transaction processing commands that are necessary extensions of the WS-BPEL language. The aim of the Concurrency Control Module is to manage the concurrent execution of business processes. The Transaction Commiter coordinates actions associated with the start of transaction, transaction committing, compensation of transaction, creating savepoints and compensation to savepoints. During coordination it takes into account declared properties of transactions.

The most important system components are described below.

4.1. The Transaction Definition File

As mentioned above, the Transaction Definition File is a definition of the business process expressed in BPEL language. This choice was made due to high popularity of BPEL and the existence of many servers, which allow for the execution of business processes expressed in that language.

Since the WS-BPEL language does not contain commands for the processing of transactions, we have proposed some WS-BPEL language extensions. Each of these extensions is a language construction, which can have a different form, dependent on the business process server, which is used. For example, in the case of Apache ODE 2.0 one can use the available extensions which have the form of insertions placed in the process definition in tags< `bpel` : `extensionActivity` >.

To enable the properties, described in Section 2, in the business processes we have defined the following extensions to BPEL language:

- *beginTransaction* – the task of this extension is to initiate the transaction, with declaration, whether the successful execution of the transaction as a child transaction is necessary for the success of the parent transaction, and whether the commitment of the transaction is dependent on the parent transaction commitment.
- *preInvoke* - the task of this extension is to inform the Transaction Commiter about the Web service call, and declare whether the successful execution of the Web service call, confirmed at the voting phase of 2PC is required for the success of the transaction (this is the only extension which has system and not business meaning).
- *savepoint* - the purpose of this extension is to create a savepoint, which is an indicating point within a transaction and allows to make a partial (to this point) compensation of a business transaction.
- *rollback* - initiates the compensation (rolling back) of the whole transaction or only of actions taking place after the savepoint declaration.
- *commit* - its task is to perform commitment and final completion of the transaction.

Of course, each extension requires implementation of appropriate system actions. In our solution they are executed by the Transaction Builder Module, which extends the Transaction Manager.

4.2. The Transaction Definition Module

The task of the Transaction Definition Module is to enable the end-user to define the business process with the usage of transaction processing commands. After analysing different possibilities we decided to use the Eclipse platform. It is very popular at the moment, allowing for the definition of the business process expressed in BPEL (Eclipse BPEL Designer) and, most importantly, it has good extension capabilities. The plug-in, which we created, greatly simplifies the definition of extensions of BPEL language (transaction processing commands) transforming the editor of BPEL process, functioning within the Eclipse platform, into a Transaction Definition Module. The definition of these extensions can be performed visually.

4.3. The Transaction Manager

The Transaction Manager is the BPEL server, which performs processes defined in the WS-BPEL language. In addition to performing actions coming directly from the WS-BPEL language instructions, due to extensions of the WS-BPEL language and their implementation, it also cooperates with other modules that are parts of the Transaction Coordinator, like the Transaction Commiter, for example. Each extension in the definition of BPEL process, according to their semantics, triggers appropriate actions which are an implementation of the extension. Thanks to this it is possible to perform essential additional actions within the business process, for example actions that control and coordinate the execution of the process. From the point of view of architecture, extensions are processed by the Transaction Builder Module. For a better readability of this article the Transaction Builder Module has been discussed after the introduction of the modules it cooperates with.

4.4. The Transaction Commiter

The Transaction Commiter has been implemented based on *WS-Coor*. Its task is the coordination of transaction processing in: activation of the transaction, registration of savepoints, approval of transaction, partial rolling back of transaction, full rolling back of transaction.

From the perspective of *WS-Coor*, the Transaction Commiter serves as a coordinator and, therefore, provides the following services:

- Activation service – creates the so called context of coordination, which serves as a transaction ID for all participants used within the particular instance of transaction executed within the business process. The activation service, which operates within the Transaction Commiter is requested by the Transaction Manager, which receives the context of coordination as a result. The context of coordination is used by the Transaction Commiter at the registration as a participant of activity. Furthermore, the context of coordination is passed on (in the headers of messages) to Web services invoked within transaction, thus they can also register as participants of the same coordinated activity.
- Register service – The task of this service is to register the participant of the activity on the basis of his context of coordination. Participants of the activity may play different roles in the process. For the Transaction Commiter the active participants are the Transaction Manager, which executes the Transaction Definition File, and services which are invoked based on the Transaction Definition File.

In addition, the Transaction Commiter uses protocol services cooperating with participants in the transaction based on the protocols, described below.

4.5. Protocols of the Transaction Commiter

Within the Transaction Coordinator individual modules work together using specific protocols. These protocols depend on the type of coordination, which is set when the context of the coordination is created by the activation service.

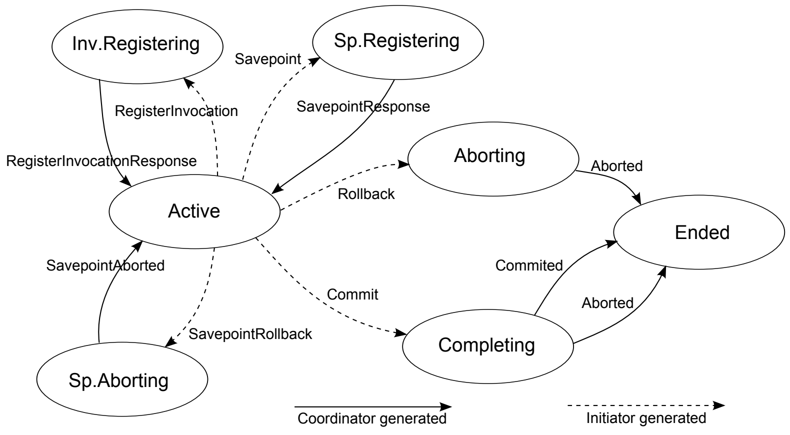

Figure 2. The *EnhancedCompletion* protocol

In the case of the Transaction Commiter two protocols are used. The first one, *EnhancedCompletion*, is used for cooperation between the Transaction Commiter and the Transaction Manager, which is the initiator of the activity. The *Enhanced2PC*, the second protocol, is used for cooperation between the Transaction Commiter and the other participants of the activity – service components and Transaction Commiters, which coordinate child transactions. Each of the protocols, its messages, their meaning and a state-chart diagram resulting from the sent messages, is introduced below.

4.5.1. The *EnhancedCompletion* protocol

The *EnhancedCompletion* protocol, whose state-chart diagram is shown in Fig. 2, is used in communication between the Transaction Coordinator, which controls the execution of transactions, as initiator of the activity, and the Transaction Commiter as the coordinator. The *EnhancedCompletion* protocol is an extension of the *Completion* protocol (see Little, 2009) with the messages, that provide additional functionality. Messages within the *EnhancedCompletion* protocol and their meaning are as follows:

- *RegisterInvocation* – sent by the initiator of the activity. Before every Web service call the initiator sends an information about it to the Transaction Commiter – *RegisterInvocation* message. Based on the information about the invoked Web service, the coordinator sends a question to the Compensation Repository (described in Section 4.6) about the character and possible compensation actions for the Web service call. After receiving an answer the coordinator writes this information in the rollback log and confirms to the initiator the receipt and registering of this information using the *RegisterInvocationResponse* message. The Transaction Coordinator can invoke the Web service only after receiving this message.

- *RegisterInvocationResponse* – sent by the Transaction Commiter as an answer to the *RegisterInvocation* message. It confirms the registration of the Web service call in the rollback log.
- *Savepoint* – sent by the initiator of the activity, using the Transaction Commiter to register the appearance of a savepoint within the transaction in the rollback log. Savepoints allow for making a partial (to this point) compensation of a transaction.
- *SavepointResponse* – sent by the Transaction Commiter in reply to the *Savepoint* message. Confirms the registration of the savepoint in the rollback log.
- *SavepointRollback* – sent by the initiator of the activity, using the Transaction Commiter to initiate the procedure of a partial compensation (rollback) of a transaction - to the selected savepoint. The Transaction Commiter executes, on the basis of entries in the rollback log, compensation for those actions, which took place from the entry of the selected savepoint.
- *SavepointAborted* – sent by the Transaction Commiter to the initiator of the activity as a confirmation of a partial compensation to the selected savepoint.

The meaning of the messages: *Commit*, *Commited*, *Rollback*, *Aborted* is similar as in the case of the *Completion* protocol.

4.5.2. *Enhanced2PC* protocol

The *Enhanced2PC* protocol, whose state-chart diagram is shown in Fig. 3, has been based on the *2PC* protocol (see Freund, 2009) and is used in communication between the Transaction Commiter, as the coordinator, and the participants of an activity other than their initiator (Web services and Transaction Commiters which coordinate child transactions). Messages within the *Enhanced2PC* protocol and their meanings are as follows:

- *RegisterCompensation* – sent by the participants of an activity, to provide information to the Transaction Commiter on compensation service related to a particular Web service call. The Transaction Commiter writes this information in the rollback log (together with the information about the Web service call, which was written earlier).
- *RegisterCompensationResponse* – sent by the Transaction Commiter to the participant of an activity as a confirmation of the registration of information about compensation service.
- *Commited* – sent by the participant informs the Transaction Commiter, that the participant has committed and forgotten the transaction. Such a participant may be safely deleted from the memory of the coordinator. It should be noted that in contrast to the *2PC* protocol *Commited* message can be sent to the Transaction Commiter, also before the voting phase of *2PC*. For example, this situation occurs in the case of participants functioning as Transaction Commiters for independent child transactions.

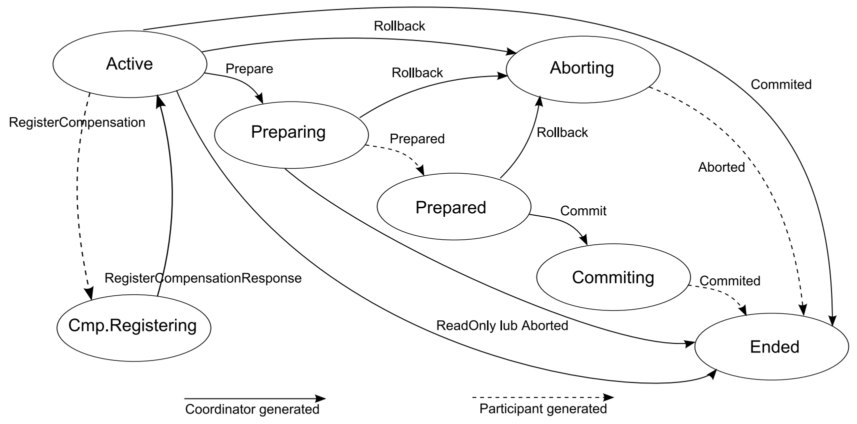

Figure 3. The *Enhanced2PC* protocol

- *Aborted* – sent by the participant, informs the coordinator that the participant has aborted and forgotten the transaction.
  Like in the case of the *Commited* message, *Aborted* can also be sent to the Transaction Commiter before the start of the voting phase of *2PC*.

The meaning of the messages: *Prepare*, *Prepared*, *Commit*, *Rollback*, *ReadOnly*, *Reply* is similar as in the case of the *2PC* protocol.

4.6. The Compensation Repository

The Compensation Repository, which is part of the Transaction Commiter, has been implemented as support service. Its task is to extend the functionality of the Transaction Coordinator with the possibility of using Web services, which do not cooperate with the Transaction Commiter according to *Enhanced2PC* protocol. Web service, together with their compensation generator, can be registered in the Compensation Repository. Thanks to this the Compensation Repository can replace the Web service in cooperation with the Transaction Commiter and on their behalf provide information on compensation actions corresponding to the particular Web service call.

The role of the Compensation Repository can be performed by other, external and independent from the Transaction Commiter, registries of Web services, if they have the required functionality.

4.7. Concurrency control mechanism

Within the Transaction Coordinator a concurrency control mechanism has been implemented. It is based on the Concurrency Control Module and Local Consistency Managers, which have been designed and implemented in the framework of our approach. It should be noted that the solution we have used was significantly inspired by the idea presented in Alrifai (2006). The most important features of this solution are: using an optimistic approach, in which the verification of the correctness of the transaction occurs during the commitment of transaction, involvement of service providers, using *WS-Coor* specification, and also preserving security of information about business processes. The optimistic approach is preferred in cases of processing of long-term business processes, in which, apart from tasks executed automatically, there are tasks that need the reaction of the end-user. Furthermore, this approach works well in situations when the resources used in the process can not be fully controlled by the process, and that is the case in Web-services based environments (see Krafzig, 2004). Service providers, in this solution, which is based on the *WS-Coor* specification, play the role of participants in the coordinated activity. Their role is to maintain compatibility matrix of provided services, registration of their usage by the particular transactions, updating of precedence graph and cooperation with the coordinator (Concurrency Control Module) during committing of transaction. The task of the service providers is not to allow to commit transaction when active preceding transactions still exist, or they were rolled back. Using service providers for the above purposes is reasonable, because they have the best knowledge about provided Web services and the relations between them. Another feature of our solution is preserving security of information about business processes. Many Web services are usually used by business processes, often belonging to more than one service provider. Information exchanged between the process and the Web service can be important for the organisation, within which the process is executed, therefore they can be confidential. Information sent between the business process and the Web service can not be passed on to others and this does not only concern the message content, but also, for example, the use of a Web service by the process. As the main idea of our solution is available in Alrifai (2006), only the modules used in our implementation of concurrency control mechanism have been briefly characterised below.

The Concurrency Control Module – from the point of view of the *WS-Coor* specification it plays the role of the coordinator, similarly as the Transaction Commiter. The difference lies in the nature of this role. The Concurrency Control Module is responsible for the coordination of business processes within the concurrency control domain.

The Transaction Manager – the business process server, together with the Transaction Builder Module (which supports BPEL extensions). It is responsible for the creation of the context of coordination for the transaction which requires it, registering as one of the participants of an activity, passing on the context of coordination to invoked Web services and communicating with the coordinator according to the established protocol. The mentioned actions result from the concurrency control mechanism of concurrent realisations of business processes.

The Local Consistency Manager – from the point of view of the *WS-Coor* specification it is a participant of an activity. On the side of each service provider, which provides Web services that are in conflict with each other, a single instance of the Local Consistency Manager is installed. There is also the compatibility matrix, which is used by the Local Consistency Manager during detection of conflict of service calls. When the Web service is requested, for the particular transactions, the Local Consistency Manager updates the local precedence graph, on the basis of the compatibility matrix. When a commitment of transaction is performed, the Local Consistency Manager cooperates with the coordinator to ensure the correct execution of business processes.

4.8. Transaction Builder Module

The Transaction Coordinator, which has a modular design, uses two modules, which act as the coordinators, from the perspective of the *WS-Coor* specification – Transaction Commiter and Concurrency Control Module. Both modules coordinate the same transactions. Each of them has a clearly defined goal. The Transaction Commiter ensures the fulfilling of declared properties of the transactions (optionality, dependency), compensates transactions, and also cooperates with the participants of an activity during commitment of transaction. The Concurrency Control Module ensures consistent realisation of concurrent transactions. Each of these modules can work separately, ensuring the appropriate properties of transactions. However, if both modules operate within one environment of the Transaction Coordinator, cooperation is needed. All the messages that initiate the next phases of transaction processing are sent by the Transaction Builder Module (which operates under the Transaction Manager), conform to the *EnhancedCompletion* and *TransactionFinalizationProtocol* protocols, which interprets the extensions of BPEL language found in the Transaction Definition File, and according to their semantics takes appropriate actions. The Transaction Builder Module is a participant of an activity (on behalf of the Transaction Manager) both for the Transaction Commiter and the Concurrency Control Module. Therefore, in a natural way, cooperation between the coordinator modules occurs with its mediation. The solution is reasonable, because the Transaction Builder Module, basing on the definition of transaction knows what coordination mechanisms in the specific cases are needed.

The Transaction Builder Module is responsible for the execution of five transaction processing commands (extensions of BPEL): *beginTransaction*, *preInvoke*, *commit*, *rollback* and *savepoint*, which can be placed in the definition of the business process (the Transaction Definition File). A detailed description of the use of these commands, in a situation when both coordination modules are being used, is presented in Jankiewicz (2010).

5. Research scenarios

After the implementation of the Transaction Coordinator, functional and performance tests were conducted. Their goal was to confirm the accuracy of assumptions, correctness of the project and finally the stability and correctness of implementation. They were related to two basic issues: complexity of business processes, and concurrent control of business processes. They included, among others, the following functional scenarios: verification of the correctness of execution of complex business processes (F.1 scenario), verification of the correctness of concurrent execution of business processes (F.2), verification of the correctness of cooperation between the coordinator modules (F.3). Furthermore, they included one of the following performance scenarios: the impact of coordination mechanisms on the system throughput, particularly with the use of nested transactions (W.1), the impact of the use of complex business process model on system throughput (W.2), the impact of the use of concurrent control mechanisms of execution of business processes on system throughput (W.3).

These test scenarios were performed, a and their results confirmed the correctness of the Transaction Coordinator. Due to limited space, the results of two performance scenarios, W.1 and W.2, are described in detail.

5.1. The impact of coordination on the system throughput (W.1)

The processing of any business process, which uses the control commands, associated with transaction processing, leads to the need of execution of appropriate actions. These actions include cooperation of the Transaction Manager with modules acting as coordinators. Each coordination introduces an additional overhead. In our case this is mainly related to the messages exchanged during coordination.

Within the scenario analysing the impact of coordination mechanisms on system throughput the time of realisation of business processes was examined. Business processes differed in the way of using and the number of transactions. Each business process consisted of 10 Web service calls. The following types of business processes were examined:

- business processes that did not use transactions, they were not coordinated at all and therefore they acted as a reference point,
- business processes that used only parent transactions (between 1 and 5 parent transactions),
- business processes that used multilevel transactions (2, 3 levels).

In the results presented the following types of business processes were taken into account:

(0) – business process that does not use transactions and therefore does not introduce any overhead of coordination, with 10 requests of two different Web services,

(1) – business process with one transaction, which has 10 requests of two different Web services,

(2) – business process with two transactions, each transaction including 5 requests of two different Web services,

(3) – business process with three transactions, the two first transactions had 3 requests of two different Web services, the last had 4 requests of two different Web services,

(5) – business process with five transactions, each of the transactions had 2 requests of two different Web services,

(2(1)) – business process with two parent transactions, each of the parent transactions had 2 requests of two different Web services and one child transaction, each of the child transactions had 3 requests of two different Web services,

(1(2(1))) – business process with a one parent transaction, the parent transaction had 2 requests of two different Web services and two the first level child transactions, each of the first level child transactions had 2 requests of two different Web services and one the second level child transaction, each second level child transaction had 2 requests of two different Web services.

In the tests the response time of the Web services was changed, simulating delays resulting from the nature of the business process. The coordination included both the Transaction Commiter and the Concurrency Control Module.

As expected, the overhead, associated with the coordination, depends on the way the transaction is used by the business process. Taking into account that each of the examined processes had the same number of Web service calls, the overhead did not depend on other factors. Tests showed that the overhead associated with coordination is constant for particular types of processes and does not depend on the response time of Web services. The average coordination time for the particular types of processes is shown in Table 1.

Table 1. The average coordination time for the particular types of processes

Process type	The average time of coordination (sec.)	Number of sent messages
(0)	0.0	22
(1)	2.8	110
(2)	4.1	134
(3)	5.5	160
(5)	8.6	212
(2(1))	4.6	198
(1(2(1)))	3.6	236

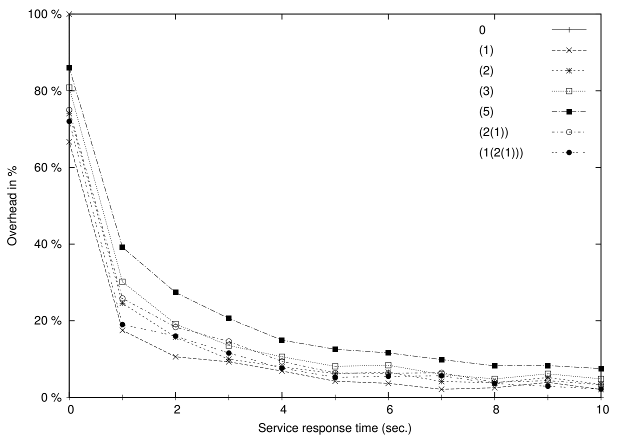

Figure 4. Percentage overhead of coordination

When we evaluate the average coordination times it should not be forgotten that the Transaction Coordinator is intended for the execution of business processes, which by their nature are long-running. The graph in Fig. 4 in a clear way reflects the real business nature of the impact of coordination mechanisms on system throughput. In the case of process **(5)** the overhead associated with the coordination falls below 10% of the execution time, at the response time of Web services around 7 seconds. This leads to the conclusion that from a business point of view the impact of coordination mechanisms on the system throughput does not really matter, if we consider the execution of process, whose duration is measured in days or weeks.

5.2. The impact of the use of a complex business process model on system throughput (W.2)

When business processes have optional components, their failure does not affect the result of the process or the parent component. This gives increased chances of success of the business process. Introduction of a complex business process model to the Transaction Coordinator enabled in a declarative way to specify optional components in the form of optional child transactions. We wanted to check in what degree the use of optional child transactions will increase system throughput, assuming that some of the child transactions may fail.

Within the scenario we examined a series of 10 business processes, each business process having five parent transactions. Each parent transaction had two independent Web service calls and five child transactions with two Web service calls. To create this series of processes we used business process generator specifically created for this purpose. It allowed for parameterisation of the probability of success of the child transaction and the probability that the child

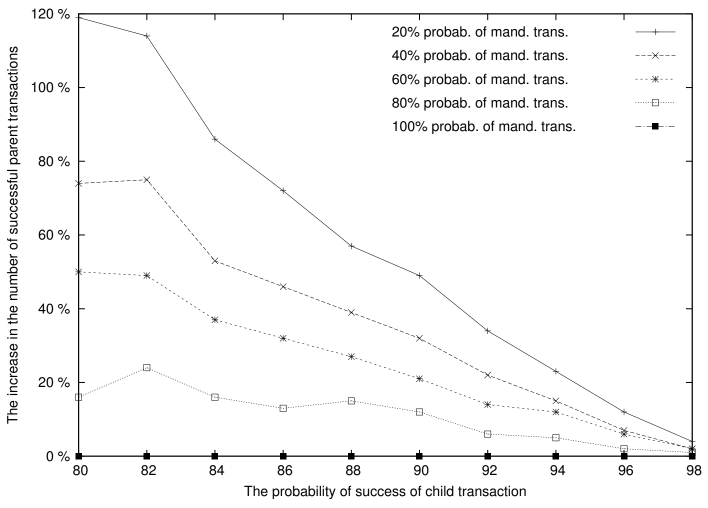

Figure 5. The increase in the number of successful parent transactions

transaction will be optional. We have examined the following measures:

- the number of successful transactions,
- the number of failed transactions,
- the number of successful processes,
- the number of Web service calls,
- the number of compensations.

Some of the results are shown in Figs. 5 and 6.

Even partial use of optional child transactions increases the chances of success of the parent transactions, and so of business processes. Fig. 5 shows the percentage increase of successful transactions in relation to a series of business processes no using optional child transactions. For example, for a series of business processes that used 60% of optional child transactions and probability of success of the child transaction reached 80%, the number of successful parent transactions was greater by 51% compared to the series in which all child transactions were mandatory.

Each failure of the parent transaction forces the compensation of Web services requested within that transaction and also those Web services, which were requested within all the dependent child transactions. Each compensation is an additional system load, which compensates the effects of earlier actions, also entailing some load. The effect of the compensation is therefore overall decrease of system performance. Fig. 6 shows the number of compensations depending on the probability of success of the child transaction for different probability of the use of mandatory child transactions. For a series of business processes

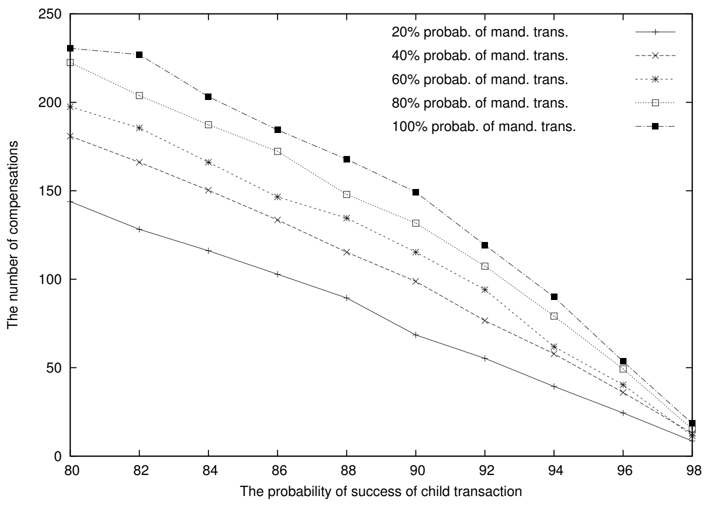

Figure 6. The number of compensations

using only mandatory child transactions with success probability of the child transaction at 80%, the number of necessary compensations was on average 231. Therefore, because each parent transaction had two Web service calls, and every child transaction also had two, there were 600 Web service calls in a series of ten business processes. This gives more than 38% of compensated calls. If the use of mandatory child transactions was limited to 60%, then for 80% probability of success of the child transaction the number of compensations will be limited by 18%, to 197.

Hence, the use of optional child transaction increases in system throughput. Of course, the optional parts of the business processes can be designed without transactions, and without the Transaction Coordinator. However, this requires a large programming effort and leads to a situation in which the system elements of the business process interweaves with the business elements. Our solution requires only appropriate declarations within the definition of business process extensions.

6. The summary and directions for further work

The article introduces the Transaction Coordinator environment. The goal of the Transaction Coordinator is to simplify the definition phase of even very complex business processes and to enable the use of commands in control of transaction processing. These commands separate system actions, taken by the environment automatically on the basis of the declared properties of individual transactions from business actions, defined by the creator of the process. During the work on the Transaction Coordinator its architecture was developed as well as protocols for communication between different modules. In addition, all the planned modules where implemented, and, to confirm the correctness of assumptions, a series of experiments were conducted. It should be noted that the objectives of the project have been achieved. This was confirmed by numerous tests based both on functional and performance research scenarios.

Further research on the Transaction Coordinator will be conducted in two directions. First, we will develop mechanisms to detect deadlock, involving more than one service provider. Although in Alrifai (2006) a solution has been proposed, but it works properly only for simple and special cases. In addition, the operational version of the Transaction Coordinator will be prepared, to be used in real applications.

References

Alrifai, M., Dolog, P. and Nejdl, W. (2006) Transactions Concurrency Control in Web Service Environment. *ECOWS '06: Proceedings of the European Conference on Web Services.* IEEE Computer Society, 109–118.

Alves, A. et al. (2009) *Web Services Business Process Execution Language.* OASIS Standard, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

Bernstein, P.A., Hadzilacos, V. and Goodman, N. (1986) *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Longman Publishing Co., Inc. Boston.

Choi, S. et al. (2005) Maintaining Consistency Under Isolation Relaxation of Web Services Transactions. *WISE*, **LNCS 3806**, Springer, 245–257.

Feingold, M. and Jeyaraman, R. (2009) *OASIS Web Services Coordination Version 1.2.* OASIS Standard, http://docs.oasis-open.org/ws-tx/wscoor/2006/06

Freund, T. and Little, M. (2009) *OASIS Web Services Business Activity Version 1.2.* OASIS Standard, http://docs.oasis-open.org/ws-tx/wsba/2006/06

Garcia-Molina, H. and Salem, K. (1987) Sagas. *ACM SIGMOD Record,* **16** (3), 249–259.

Jankiewicz, K., Kujawiński, K., Mor, M. and Morzy, T. (2010) *Mechanizmy transakcyjności w kompozycji usług w złożone procesy biznesowe* (in Polish). Technical report. TR-ITSOA-OB2-5-PR-10-01. Institute of Computing Science, Poznań University of Technology.

Krafzig, D., Banke, K. and Slama, D. (2004) *Enterprise SOA: Service-Oriented Architecture Best Practices.* Prentice Hall, The Coad Series, Upper Saddle River, NJ, USA.

Lieberman, B. (2006) *SOA transaction management.* IBM developerWorks.

Little, M. and Wilkinson, A. (2009) *OASIS Web Services Atomic Transaction Version 1.2.* OASIS Standard, http://docs.oasis-open.org/ws-tx/wsat/2006/06

McGovern, J., Sims, O., Jain, A. and Little, M. (2006) *Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

McGovern, J., Tyagi, S., Stevens, M. and Mathew, S. (2003) *Java Web Services Architecture.* Morgan Kaufmann.

Younas, M., Li, Y., Lo, C.-C. and Li, Y. (2006) An Efficient Transaction Commit Protocol for Composite Web Services. *AINA 2006: Proc. of the 20th International Conference on Advanced Networking and Applications.* IEEE Computer Society, 591–596.