Confirmation: a Solution for Non-compensatability in Workflow Applications

Chengfei Liu
School of Comp. Sci.
Univ. of Technology, Sydney
Sydney NS W 2009 Australia
liu@socs.uts.edu.au

Maria Orlowska Xiaofang Zhou
Dept. of CS & EE
Univ. of Queensland
Brisbane QLD 4072 Australia
{maria, zxf}@csee.uq.edu.au

Xuemin Lin
School of Comp. Sci. & Eng.
Univ. of New South Wales
Sydney NSW 2052 Australia
lxue@cse.unsw.edu.au

The notion of a compensation is widely used in advanced transaction models as means of recovery from a failure. Similar concepts are adopted for providing "transaction-like" behavior for long business processes supported by workflows technology. Generally, designing a compensating task in the context of a workflow process is a non-trivial job. In fact, not every task is compensatable. This work contributes to the study of the non-compensatability problem.

A compensating task C of a task T semantically undoes the effect of T after T has been committed. For example, the compensating task of a deposit is a withdrawal. For a task to be compensatable, it must satisfy two conditions.

- forcibility: The compensating task of the task must be forcible. In other words, after the task commits, the execution of its compensating task is guaranteed to succeed by the application semantics.
- relaxation of isolation: The isolation requirement of the shared data resources which the task may access must be relaxed. This relaxation is required as the purpose of introducing compensation is to avoid long-duration waiting, otherwise, compensation may become useless.

In this work, we carefully investigate the properties of shared resources and tasks which may be performed on these resources. As all its invoked operations must be compensatable as well if a task is compensatable, we only discuss the compensatability of operations defined on shared resources. For most non-compensatable operations, their reverse operations are only conditionally non-forcible. i.e., the reverse operation of a non-compensatable operation may fail only when *undesired condition* is reached. Based on this, it is ideal to provide a semantic level mechanism which can be used to guarantee that the undesired condition will never be reached. For this purpose, we propose a new concept called *confirmation.*

A confirmation is defined as part of an operation. It specifies the extra work which needs to be done (confirmed) after the normal part of the operation commits. In other words, the work of an operation is divided into a normal part and an extra part, if needed. The normal part of an operation is performed when the operation is invoked, while the extra part, as specified by the confirmation, is executed later.

To ensure that the undesired condition will never be reached, we can put unsafe work of an operation (e.g., deposit) into its confirmation part and delay the execution of this part until a safe time later on, say, after an invoking workflow instance succeeds in its execution. At that time, changing the value of the undesired condition by other operations (e.g., withdraw) will not cause any problem because the compensation of the operation invocation is no longer needed. As a result, the normal part of an operation can be always compensated before the execution of the confirmation part of the operation. By using 'a confirmation, we are able to modify some originally non-compensatable tasks so that they become compensatable.

Note, the normal part and confirmation part *areforward* parts of an operation. Unlike the compensation *backward* part) of an operation, if the forcibility of either the normal part or the confirmation part cannot be guaranteed by application semantics, it will not leave any problem as the invoking workflow can always choose to fail or try a contingency plan.

In facilitating compensation and confirmation in a transactional workflow environment, *compensate* and *confirm* statements should be allowed to put in a workflow specification. This is similar to including *abort* and *commit* statements in a transaction. Upon success (or a safe stage) of a workflow instance, the confirmation parts of all executed operations, if defined, are executed; Upon failure of a workflow instance, the compensation parts of all executed operations, if defined, are executed. The difference between a workflow scenario and a transaction scenario is that execution of *compensate* and *confirm* statements is application behavior, while execution of *abort* and *commit* statements is system behavior. In this light, the pair of concepts *confirm/compensate* can be regarded as *semantic commit/abort.*

A framework has been proposed to incorporate both confirmation and compensation into transactional workflows.

0-7695-0071-4/99 $10.00 © 1999 IEEE

Authorized licensed use limited to: UNSW Library. Downloaded on August 10, 2009 at 06:59 from IEEE Xplore. Restrictions apply.