On the Semantics of Distributed Compensations with Interruption

Roberto Bruni[1], Anne Kersten[2], and Ivan Lanese[3]

[1] Department of Computer Science, University of Pisa, Italy
[2] IMT Lucca, Institute for Andvanced Studies, Italy
[3] Lab. FOCUS, University of Bologna/INRIA, Italy

Compensations is a well-known and widely used mechanism to ensure the consistency and correctness of long-running transactions in the area of databases. More recently, several compensable workflow languages and calculi emerged in the area of business process modelling, service-oriented and global computing to provide the necessary formal ground for compensation primitives like those exploited in orchestration languages like WS-BPEL. The focus of this work is on the semantics of the workflow based calculus Sagas [2] and its compensation policy for parallel branches. The choice of the right strategy allows the user to prevent unnecessary actions in case of an abort. An optimal formalization of this policy is still an open problem that we want to solve.

In the past different policies have emerged. A thorough analysis is presented in [1] by comparing the Sagas calculus with compensating CSP [3] (cCSP). Both Sagas and cCSP focus on a core set of operations, namely compensation pairs $A \div B$ for two basic activities A and B, sequential composition of processes $P; Q$ as well as the parallel composition $P|Q$ and a transaction scope $\{\![P]\!\}$ (called saga). The key idea is that for a pair $A \div B$ the successful execution of A installs the compensation B, to be executed for "undoing" A in case the transaction is aborted later on. In case of sequential composition, the latest actions that have been executed are the first to be compensated, hence compensations are unfolded in the reverse order of installation. The different strategies emerge from different approaches to handle compensations in a concurrent setting, by taking into account two main aspects. One aspect concerns the interruption of siblings in case of an abort(*interruption* or *no interruption*). The other depends on the managing of whether compensations are started at the same time (*centralized*) or siblings can start their compensation on their own (*distributed*). The relation between the different policies is displayed in Fig. 1.

As it turns out, none of the four originally defined semantics is entirely satisfactory. Strategies one to three are too restrictive: it is important to have the possibility to stop a sibling branch when a problem occurs and to activate compensations as soon as possible, because typically compensations have a cost. Without interruption (cases one and two) sibling branches finish their execution anyway, even though they will have to compensate. In the centralized case three, branches might have to wait until they are allowed to continue together with their siblings. The fourth strategy on the other hand is irrealistic, it allows a kind of guessing mechanism where a branch may start its compensation even though the error has not yet occurred. This holds as well for an asynchronous

Fig. 1. Compensation policies (arrows stand for trace inclusion)

setting. An optimal, realistic semantics should be more "permissive" (in the sense of allowing more traces) than strategies two and three but less than four.

A recent fifth approach is presented in [5], but it is not expressed at the same abstract level of detail, because its formalization relies on spurious mechanisms closer to the implementation level. Moreover the fifth policy is not distributed, as the order of the execution of compensations depends on the total order of activities in the forward flow.

In this paper, we propose a new, optimal semantics for parallel Sagas with interruption. Technically, the result is achieved via an encoding of parallel Sagas into a calculus with dynamic compensations inspired by [4]. It uses message passing for sending an interrupt to parallel branches. Thus compensations may only be started after the error actually occurred.

To clarify the differences between the different policies discussed above, we want to present different sets of traces obtained for a toy example. We consider the term $\{[(A \div A'; B \div B')|(C \div C'; \mathit{abort} \div \mathit{skip}]\}$. It may stand for example for a workflow for ordering products, where A stands for choosing the product, the activity B for filling in an adress form while C is the credit card check, that may currently be unavailable and therefore fails. Each compensation would be the sending of failure notification by email.

For case one, centralized compensation without interruption, the resulting set of traces is $S_1 \equiv (AB|||C)(B'A'|||C')$ (where $|||$ stands for the interleaving of traces and juxtaposition stands for concatenation), i.e., all branches are fully executed forward and only then their (interleaved) compensation is started.

For case two, distributed compensation without interruption, the set of traces is $S_2 \equiv (AB|||CC')B'A'$. The first branch still finishes its execution, nevertheless C may already start its compensation as soon as it aborts.

For case three, we have $S_3 \equiv CC' \cup (A|||C)(A'|||C') \cup (AB|||C)(B'A'||C')$, i.e., the branch for the activities A and B may be interrupted, however compensations are started only when each branch is ready.

The fourth strategy, distributed interruption, is the most liberal one. It allows the following set of traces: $S_4 \equiv CC' \cup AA'|||CC' \cup ABB'A'|||CC'$. Note that means that B' can also be executed before C, i.e.before the error occured.

In our new semantics: $S \equiv S_3 \cup (CC'AA') \cup (AB|||CC')B'A'$.

References

1. Bruni, R., Butler, M.J., Ferreira, C., Hoare, C.A.R., Melgratti, H., Montanari, U.: Comparing Two Approaches to Compensable Flow Composition. In: CONCUR'05. LNCS, vol. 3653, pp. 383–397 (2005)
2. Bruni, R., Melgratti, H., Montanari, U.: Theoretical foundations for compensations in flow composition languages. In: POPL'05. pp. 209–220. ACM (2005)
3. Butler, M.J., Hoare, C.A.R., Ferreira, C.: A Trace Semantics for Long-Running Transactions. In: 25 Years CSP. LNCS, vol. 3525, pp. 133–150 (2004)
4. Lanese, I., Vaz, C., Ferreira, C.: On the expressive power of primitives for compensation handling. In: ESOP'10. LNCS, vol. 6012, pp. 366–386 (2010)
5. Lanese, I., Zavattaro, G.: Programming Sagas in SOCK. In: SEFM'09. pp. 189–198. IEEE (2009)