Modeling of Compensation in Long-Running Transactions

Rebwar Mala Nabi
Technical College of Informatics
Sulaimani Polytechnic University
Sulaimani, Iraq

Rebaz Mala Nabi
Technical College of Informatics
Sulaimani Polytechnic University
Sulaimani, Iraq

Sardasht M-Raouf Mahmood
Statistics and Computer Department
University of Sulaimani
Sulaimani, Iraq

Rania Azad Mohammed
Computer Science Institute
Sulaimani Polytechnic University
Sulaimani, Iraq

***Abstract*—nowadays, the most controversial issue is transaction in database systems or web services. Specifically, in the area of service-oriented computing, where business transactions always need long periods of time to finish. In the case of a failure rollback, which is the traditional method, it will not be enough and not suitable for handling errors during long running transactions. As a substitute, the most appropriate approach is compensation which is used as an error recovery mechanism. Therefore, transactions that need a long time to complete are programmed as a composition of a set of compensable transactions. This study attempts to design several compensation policies in the long running web transaction especially when the transaction has parallel threads. Meabwhile, one thread in sequence steps of the transaction may fail. This paper also describes and models many different ways to compensate to the thread. Moreover, this study proposes a system to implement creating long running transactions as well as simulating failures by using compensation policies.**

Keywords—transaction; compensation; long-running transaction and interruption

I. INTRODUCTION

Over the past decades, business transactions have become incredibly important and compound. It is usually refered to as coordinations and communications between multiple partners. In this case, faults are possible at any stage of the transactions. In standard ACID transaction, (with properties of Atomicity, Consistency, Isolation, and Durability), solving and handling faults are done by using a rollback mechanism to provide all or nothing atomicity for transactions [1]. However, rollback is not always satisfactory, especially for transactions needing long life of the time, as widely known as long running transactions (LRT). LRTs usually involve more than one agent. It can be seen that handling faults or errors are difficult and critical, particularly when several partners are involved. Check-pointed is impossible in the LRTs due to their nature, e.g. an email that was sent cannot be unsent. Therefore, LRTs need a comprehensive and separate mechanism to solve such problems. In order to terminate such problems the system need to be designed as a compensation mechanism for those actions that cannot be undone.

Compensation as described in [1] is taken from recovery faults in a business transaction as an action. Consider bookshop as an example, a user buys some books from an online book shop. The system charges the customer's account for the payment of the selected books. Meanwhile, the store of the bookshop knows that one or more books are not on hand at that time. So, to compensate the customer the system can refund the amount already debited and also notified the customer about the situation. Based on this scenario, the importance of the compensation is more reasonable than traditional database rollback. It can be argued that compensations are imperative in terms of handling faults in the long-running transaction. Compensations in LRT are set up for every committed activity.

Recently, many policies have been produced and proposed which used to define approaches of modelling LRTs such as Sagas [3], StAC [4, 5], CSP [6], πcalculus [7, 8], Join calculus [9]. There are also some compensation policies as shown in figure 1. Firstly, no interruption and centralised compensation; Compensations only occur, if some transactions abort at the end of the process after all branches were executed [17, 18, 19, 20, 21]. Secondly, no interruption and distributed compensation; Parallel flows compensate as needed without others to complete [17, 18, 19, 20, 21]. Thirdly, coordinated interruption; Parallel branches are free to stop when abortion happens, but compensation is in a centralised way [17, 18, 19, 20, 21]. Finally, distributed interruption which flows are interrupted, if needed, and later on their compensation procedures can be activated independently from the rest of the flows [17, 18, 19, 20, 21]. In the case of compensation, all executed steps have to be compensated in reverse order. The system evaluates all possible compensation forms. It implies we have to compensate the executed steps more than once, each time in a dissimilar manner. For instance, if we have four executed steps, then we have to pay off the system twenty four times for example.

In this study, we model some different policies of compensation in the LRTs. It is obvious that the transaction has many parallel as well as sequence steps and ocasionally some of them may fail.

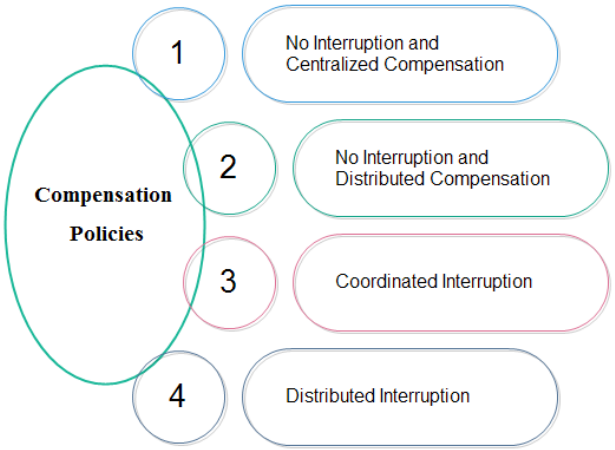

Fig. 1. Compensation Policies

Therefore, defining and applying all possible ways of compensating found to be crucial for the falsified cases. In addition, developing a system to create such transactions and impelementing it. Through using compensation policies failures will be simulated.

II. Litreture Review

Nowadays, integration between applications and processes is needed in web services on the enterprise level. Web services need a mechanism of transaction that run long-running transactions to address loosely coupled threads, instead of having traditional ACID transaction. BPML [10] by BPMI, XLANG [11] and BizTalk [12] by Microsoft, WSFL [13] by IBM, BPEL4WS [14] by OASIS, these proposals all use long running transactions to describe the activities.

In the sense of web services, interaction and coordination among various services that each one might refer to different companies involve in business transactions. Dealing with faults that occur at any level of such situations in long running transactions is essential but it is difficult. Traditional database mechanism in a long running transaction, such as rollback, is not suitable to rectify faults. For example, cancellation of hotel booking, or hiring taxi, in such cases rollback is not an option as they may need further instructions to handle faults. Usually, in the real world of the long running transaction undoing the transaction is difficult.

Ideally, the idea of compensation was introduced to recover from faults in long-running transactions. As described in [1] compensation is a mechanism to handle the error or change in the plan. When one branch goes wrong in a long running transaction, programmable compensations are to be set to compensate the parts that already completed of the transaction. The transaction concept was defined in [15] where compensation is suited with transactions that correct errors or faults of committed transactions. The later idea of the saga in [16] was defined; to describe long running transaction compensations. Transactions in the saga are divided to a chain of sub-transactions and each of them has its own compensation. Compensation of the committed sub-transaction executes when failures occur of a sub-transaction in the sequence.

There are other approaches that have been modelled in the long-running transaction and have used compensation mechanism. The πt-calculus in [22] which is inspired by BizTalk and it contains asynchronous polyadic as an extension π-calculus [23] with the transaction notion. However, the compensation is defined statically in each transaction.

The extension of Join calculus [24] is the cJoin calculus [25] with the primitives for representing transactions. The compensation method is defined statically in this calculus.

StAC language is another model in [26], which is inspired by BPBeans. The language includes the theory of the compensation pair; it is similar to the conception of sagas that is defined by Gargia-Molina and Salem [27]. In StAC, a long-running transaction is a structure of one or more sub-transaction. On the same hand, compensating CSP [28], proposed by cCSP, and Sagas calculi [29] are also based on flow composition, particularly, the embrace a centralized coordination mechanism. However, they have different compensation policies. A calculus named webπ that described by Laneve and Zavattaro [30]. It is also an extension of π-calculus with a timed transaction structure. Webπ∞ is the untimed version of webπ that was proposed by Mazzara and Lanese [31]. Although, both calculi and Webπ∞ are different in some syntax, with following different rules. Namely, nested transactions are surfaced. Therefore, nested failure do not supply by these calculi because the abortion of the sub - transaction cannot be affected by the failure of a transaction.

An extension of SOCK [33] expanded by GuiDi et al. [32], which inspired by WSDL and BPEL. The clear primitives for dynamic handler installation included in this calculus, for example, error and compensation handlers and automatic failure announcement. The correctness properties, namely expected behaviour of a scope cited by them for their calculus, the right termination upon an error, communication and the warranty correct behaviour of fault activation. Our approach is dissimilar in the sense that activation of compensations is obvious to the user. Therefore, addressing the syntax is explicitly and uncomplicated. Related to webπ∞, the only identification of the transaction is assumable by them, do not help to ensure this specialty. Another difference is that we utilize a kind system to assure soundness and activation of transaction compensations.

Gray [36] defines compensating transactions as expanding on the same idea, later saga [39], as an added layer on top of ACID transactions. This is debatable: (i) ACID transactions are impossible in the long live transaction, as in the long period of time systems, locking resources are impractical in a highly concurrent world, and (ii) nesting transactions are not supported in the ACID transaction, so working with compensation as counter transaction, transactions can be composed and nested into a saga. According to contemporary literature there are compensable transactions as shown in figure 2.

A. Atomicity

All or nothing; That is, all the changes to the state of the transaction are done, none of them happen. For example, sending money from one account to another, the atomicity

guarantees that, if a debit is made successfully from the first one, the responding later is made to the other account.

B. Consistency

When a transaction either begins and when it finishes its execution, data is in a consistent state. For example, transferring some amount from one account to another, the consistency secures that the transfer amount in both accounts at the same of each transaction.

C. Isolation

The intermediate step of the transaction is hidden to other transactions. As a result, transactions that run simultaneously seems to be serialized. If we consider the example above, the isolation feature assures that the transferred money in one or the others can be seen by another transaction, but it is impossible in both.

D. Durability

Changing to data continues and cannot be undone, once the transaction successfully finishes even in the case of failure. Again, the same example, the durability property warrants that the accounts have been changed, it will not be converted.

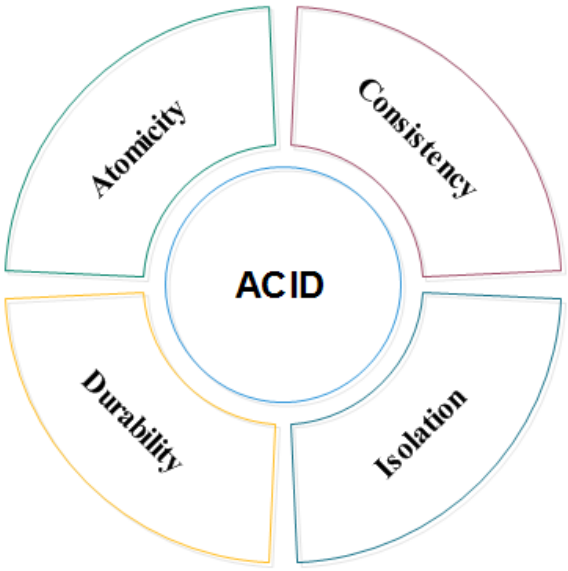

Fig. 2. ACID

III. System Design

This chapter will describe and justify the design of the system and how it satisfies the requirements. The diagram below demonstrates the intended design for the system.

As can be seen in the figure 3, the process starts by indicating the numbers of the steps that involve in the transaction digram. Afterwards, each step has a compensation step as well as having a name. Then, users can create a connection between the steps and the diagram according to the connections that have been created by them. Also, there is a command to check the connection, whether there is any incorrect connection such as loop connection and the connection between the step and itself.

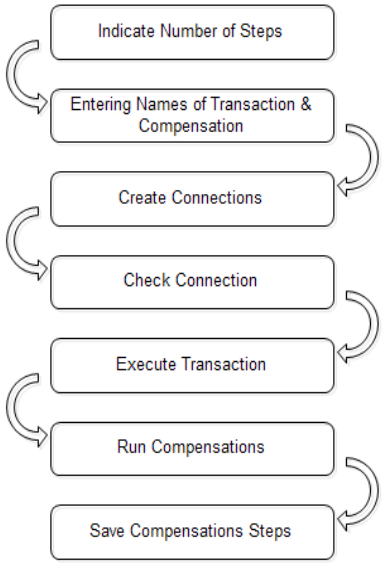

Fig. 3. System Design

After that, by that time a digram is ready to start. Once, the transaction command is pressed, the transaction starts as a forward execution. This means from the beginning to the end (Digram may contain parallel and sequence steps.).

This study demonstrates two policies regarding forward execution. They are distributed interruption and no interruption and distributed compensation. The former stops the whole execution immediately after catching the fault [17, 18, 19, 20, 21]. In contrast, the latter parallel step may continue until the completion [17, 18, 19, 20, 21]. It is unlikely forthe transaction digram expecting an error in each step of the whole transaction. Assume that the transaction digram catches he fault and now the system should compensate for these steps that already were executed. This absolutely happens in reverse order. The compensation, which is our main goal, likely to be run many times and each time compensates the executed step in a different order. More importantly, in order to remember which order they compensated, the system has to save the steps that are compensated each time.

IV. Implementation

This section in this study intends to explain the implementation of the compensation of modelling in long running web services transaction. C# programming language has been used to build a system and design a graphical user interface (GUI). The GUI allows users to choose the number of the transaction steps and preparing all the works that are needed for the system. The designed system consists of many different parts. Figure 4 shows the GUI system.

As shown in figure 4, the system contains different parts. In the beginning, the number of the transaction steps should be chosen. Then the names of the steps have to be written as well as the names of the compensations that are programmed for each step. This is used for creating dynamic diagrams which users can indicate and create a connection on demand.

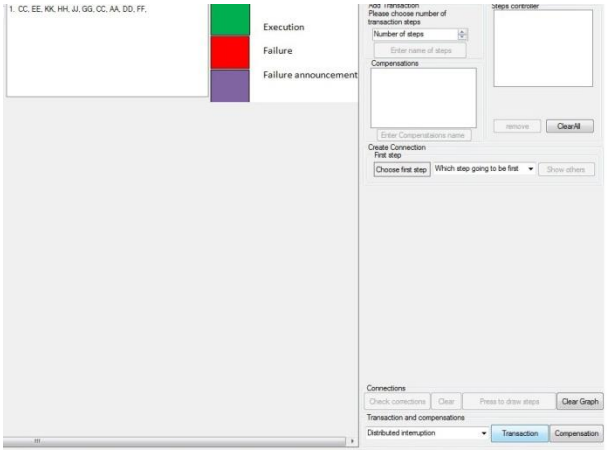

Fig. 4. Compensation of Modelling in web services LRT

Moreover, there is an option that offers two saved diagrams. Hence, users have two options, one they can create diagrams as they want, later is to choose one among these diagrams that are already saved before. Continuously, the second peace is that the users' concern with creating connections between steps. In this section system should prevent users to create incorrect connections such as loop connections or any inconvenient relations.

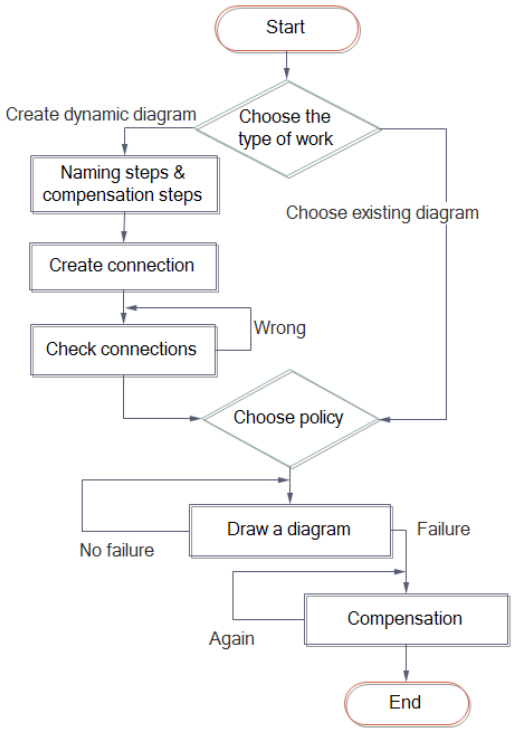

Fig. 5. Compensation of Modelling in web services LRT

V. Case Study

1) Distributed Interruption

As usual the transaction starts from the beginning of the diagram. In the figure 6, there are three parallel branches and two sub-transaction. The system runs the first step and then should randomly choose one of the three branches and then continues to the end. According to the distributed interruption policy, when the transaction gets an error, all other steps in the diagram are forced to stop the forward execution by the system. Meanwhile, the system should also set up a checkpoint in the step after the last executed steps.

For instance, according to figure 6 the system starts the transaction from A, and the form one among B, C and D randomly. The system continues until it faces a failure in the F step. Then the system immediately asks to stop the forward execution, and is ready to compensate the executed steps. Again, from the diagram, the green color declaration to the successful execution, red color indicates failure occurred and the purple colors are checkpoints as knowledge centre to the previous step that the system stopped because of failure.

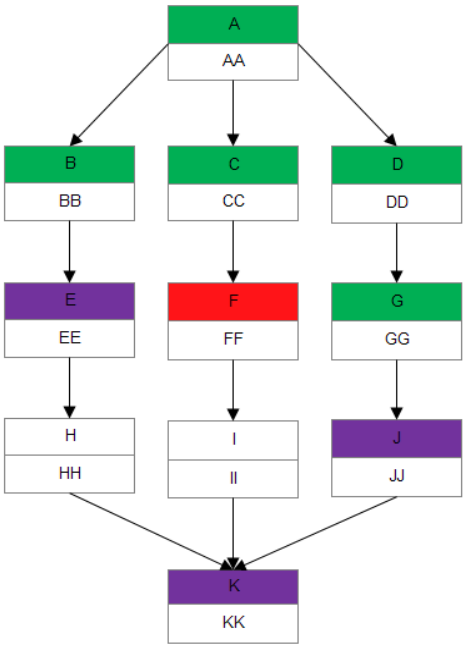

Fig. 6. Distributed Interruption

Simultaneously, in the case of failure, all steps that have been executed are likely to be compensated. Unlike the forward execution, it starts from the reverse order, from the end to the beginning. This study attempts many different simulations. This means the system compensates all executed steps in the different ways. For example, the first step that should start its compensated is the red color, then there are three options GG, CC and BB that should be compensated. Noticeably, we have a lot of probabilities to compensate. To exemplify, the set of compensation should be FF, GG, CC, BB, DD and AA, or FF, BB, GG, CC, DD and AA, or FF, CC, GG, BB, DD and AA, or FF, GG, DD, BB, CC and AA, and so on.

2) No interruption and Distributed Compensation

The transaction starts from the beginning to the end. Similar to the distributed interruption, parallel branches should randomly start the forward execution, but the only and very important point is that the difference in the failure case. Thais means when the transaction failed, the parallel branches still continue to the step that cannot be continued any longer (the step that set up the checkpoint in.). If consider figure 7 with a bit change from figure 6.

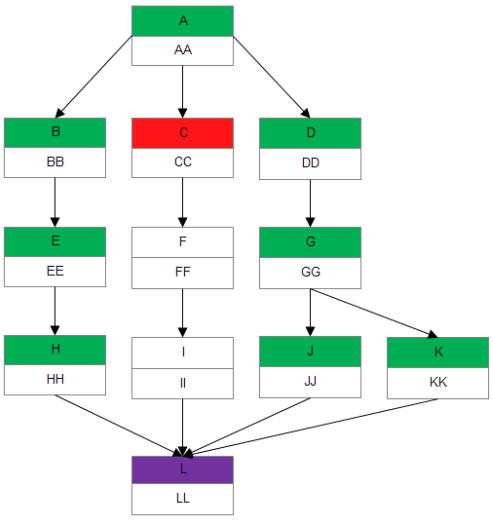

Fig. 7. No interruption and Distributed Compensation

From the figure 7, the transaction starts from step A, then B, C or D to the end. Once the transaction noticed that the failure occurred, unlike the distributed interruption, parallel branches can continue and have not been informed about the issue until reaching the step that cannot be executed (that is the purple color). As shown in the above diagram, green colors indicate the executed steps, red color means failure and the purple one is a point to inform the previous step that the system went wrong and cannot be run any more.

In the term of compensation, similar to the distributed interruption it starts from the back to the beginning. The executed steps should start their compensation as soon as the system failed. There are a lot of compensation simulations such as CC, JJ, HH, DD, EE, KK, GG, DD, BB, and AA, or CC, KK, HH, JJ, GG, EE, DD, BB, and AA, or CC, HH, KK, EE, JJ, GG, FF, DD, and AA, or CC, JJ, KK, GG, KK, HH, GG, EE, DD, BB and AA, or so on.

VI. Evaluation and Analysis

The original goal of this project was to build a system to model compensation in the long running transaction. This means when we have a diagram including the parallel and sequence steps, once one of the sequential steps fail the system should compensate to the step and to the other steps as well. It is worth mentioning that the main aim is achieved.

The system is capable of creating dynamic diagrams and users can also choose the organised diagrams. Moreover, the system allows the users to indicate the number of steps and make them being able to create connections between involved steps. Another feature is that the system can check connection. Afterwards, the diagram can be drawn easily. Once it has been done, the diagram can be transacted according to the distributed interruption and no interruption and distributed compensation. The former, the execution of the steps may stop immediately when a failure occurs. Later, the parallel step may continue to the end. It is clear that it is the forward execution. At the same time,, in the case of failure the system has to compensate the failed step and the others that are already executed. In contrast, the compensation starts from the end to the beginning. As the system programmed, the compensation mechanism should provide different ways to compensate these steps.

VII. Conclusion and Future Work

In conclusion, the complex business logic, many partners interactions today transactions require a strong framework able to handle efficiently with failures. Furthermore, because of the communication involved with various agents transactions became more and more important and longer, delivering ACID transactions unsuitable. The mechanism of compensation was introduced to transactions, enabling them to manage the new difficulties. Later, the sense of compensable transactions developed and were integrated with more complicated models concerning, amongst other aspects, parallelism, exception handling, transaction composition, and communication amongst activities. Many approaches and models have emerged, providing different solutions to the design issues involved.

In this study, the modelling of compensation in long running transaction system has been developed. The system allows users to choose between diagrams that saved or to create dynamic diagrams. Moreover, if users decide to work on the dynamic one, there are a variety of features such as giving a name to each transaction step as well as for each programmable compensation. Furthermore, some other features are applied, for exemple, creating a connection between transaction steps. Then the system can approve these connections that were made if the system found that connections are not correct, it immediately informs the users to check and change these inappropriate connections. Once, the system knows that the connections are all connected correctly, the diagram can be drawn according to the connections which have been made. Additionally, now the users have a diagram that is ready to run, by running it, the transaction starts as forward execution. In the case of failure, the system should compensate the executed steps. More importantly, by repeating the compensation, users might get a different set of compensation steps.

Overall, it can be argued that some requirements have been achieved whereas some have not. The system was developed successfully, two policies have been implemented and the compensation mechanism was performed. On the other hand, only two policies have been used which means more policies would be more efficient. Similarly,, multi-threading is reliable and efficient to design such a system, however, this system is not created by them. The main goal has achieved.

It can be identified that further researchs need to be made to improve the system more. Firstly, create a bigger diagram by increasing the number of steps or offering more samples. Additionally, create a system that allows a user to indicate all the transaction step's location not only the first step. Secondly, further research could be done to add more compensation policies to the system. In this study, only distributed interruption and no interruption and distributed compensation were used. Therefore, adding more compensation policies may increase the system's reliability and performance. Additionally, using multi-threading in order to make the system more efficient, insuring and working properly. Finally, when users

intend to use the dynamic diagram they have to move steps and order, but in the new research this can be updated and be changed to create diagrams that do not need to be organized.

REFERENCES

[1] Jim Gray and Andreas Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, 1993.

[2] Hector Garcia-Molina and Kenneth Salem. Sagas. In Umeshwar Dayal and Irving L. Traiger, editors, SIGMOD Conference, pages 249–259. ACM Press, May 27-29 1987.

[3] Roberto Bruni, Hern´an Melgratti, and Ugo Montanari. Theoretical foundations for compensations in flow composition languages. In POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 209–220. ACM Press, 2005.

[4] Michael Butler and Carla Ferreira. A process compensation language. In Integrated Formal Methods (IFM'2000), volume 1945 of LNCS, pages 61 – 76. Springer-Verlag, 2000.

[5] Carla Ferreira. Process Modelling of Business Processes with Compensation. PhD thesis, University of Southampton, November 2002.

[6] Michael Butler, Tony Hoare, and Carla Ferreira. A trace semantics for long-running transactions. In A.E. Abdallah, C.B. Jones, and J.E. Sanders, editors, Proceedings of 25 Years of CSP, volume 3525 of LNCS, London, 2004. Springer-Verlag.

[7] Robin Milner. A calculus of mobile processes. Journal of Information and computing, 100 (1): 1–77, 1992.

[8] Joachim Parrow. An Introduction to the _-Calculus, chapter 8, Handbook of Process Algebra, pages 479–543. Handbook of Process Algebra. Elsevier, 2001.

[9] C´edric Fournet and Georges Gonthier. The reflexive chemical abstract machinceand the Join-calculus. In POPL '96, 23rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 372–385. ACM Press, 1996.

[10] Business process modeling language (BPML). [www.bpmi.org].

[11] S. Thatte. XLANG: Web Services for Business Process Design. Microsoft Corporation, 2001. [www.gotdotnet.com/team/xml/wsspace/xlang-c].

[12] B. Metha, M. Levy, G. Meredith, T. Andrews, B. Beckman, J. Klein, and A. Mital. BizTalk server 2000 business process orchestration. IEEE Data Engineering Bulletin, 24 (1): 35–39, 2001.

[13] Frank Leymann. The web services flow language (WSFL1.0). Technical report, Member IBM Academy of Technology, IBM Software Group,2001. [http://www4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf].

[14] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, version 1.1., 2003. [http://www-106.ibm.com/developerworks/library/ws-bpel/].

[15] Jim Gray. The Transaction Concept: Virtues and Limitations (invited paper). In Very Large Data Bases, 7th International Conference, pages 144–154. IEEE Computer Society, 1981.

[16] Hector Garcia-Molina and Kenneth Salem. Sagas. In Umeshwar Dayal and Irving L. Traiger, editors, SIGMOD Conference, pages 249–259. ACM Press, May 27-29 1987.

[17] Bruni, R., Butler, M., Ferreira, C., Hoare, T., Melgratti, H., & Montanari, U. (2005). Comparing two approaches to compensable flow composition. In CONCUR 2005–Concurrency Theory (pp. 383-397). Springer Berlin Heidelberg.

[18] Bruni, R., Kersten, A., Lanese, I., & Spagnolo, G. (2012). A new strategy for distributed compensations with interruption in long-running transactions. In Recent Trends in Algebraic Development Techniques (pp. 42-60). Springer Berlin Heidelberg.

[19] Bruni, R., Ferrari, G., Melgratti, H., Montanari, U., Strollo, D., & Tuosto, E. (2005). From theory to practice in transactional composition of web services. In Formal Techniques for Computer Systems and Business Processes (pp. 272-286). Springer Berlin Heidelberg.

[20] Bruni, R., Kersten, A., & Lanese, I. (2010). On the Semantics of Distributed Compensations with Interruption.

[21] Lanese, I. (2010). Static vs dynamic sagas. arXiv preprint arXiv:1010.5569.

[22] Bocchi, L., Laneve, C., Zavattaro, G.: A calculus for long-running transactions. In Najm, E., Nestmann, U., Stevens, P., eds.: FMOODS. Volume 2884 of LNCS., Springer (2003) 124–138.

[23] Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press (2001).

[24] Bruni, R., Melgratti, H.C., Montanari, U.: Nested commits for mobile calculi: Extending join. In L´evy, J.J., Mayr, E.W., Mitchell, J.C., eds.: IFIP TCS, Kluwer (2004) 563–576.

[25] Fournet, C., Gonthier, G.: The reflexive cham and the join-calculus. In: POPL. (1996) 372–385.

[26] Butler, M.J., Ferreira, C.: An operational semantics for StAC, a language for modelling long-running business transactions. In Nicola, R.D., Ferrari, G.L., Meredith, G., eds.: COORDINATION. Volume 2949 of LNCS., Springer (2004) 87–104

[27] Garcia-Molina, H., Salem, K.: Sagas. In Dayal, U., Traiger, I.L., eds.: SIGMOD Conference, ACM Press (1987) 249–259

[28] Butler, M.J., Hoare, C.A.R., Ferreira, C.: A trace semantics for long-running transactions. In Abdallah, A.E., Jones, C.B., Sanders, J.W., eds.: 25 Years Communicating Sequential Processes. Volume 3525 of LNCS., Springer (2004) 133–150

[29] Bruni, R., Melgratti, H.C., Montanari, U.: Theoretical foundations for compensations in flow composition languages. In Palsberg, J., Abadi, M., eds.: POPL, ACM (2005) 209–220

[30] Laneve, C., Zavattaro, G.: Foundations of web transactions. In Sassone, V., ed.: FoSSaCS. Volume 3441 of LNCS., Springer (2005) 282–298

[31] Mazzara, M., Lanese, I.: Towards a unifying theory for web services composition. In Bravetti, M., N´u˜nez, M., Zavattaro, G., eds.: WS-FM. Volume 4184 of LNCS, Springer (2006) 257–272

[32] Guidi, C., Lanese, I., Montesi, F., Zavattaro, G.: On the interplay between fault handling and request-response service invocations. In: 8th International Conference on Application of Concurrency to System Design, IEEE Computer Society (2008) 190–199

[33] Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G: A calculus for service oriented computing. In Dan, A., Lamersdorf, W., eds.: ICSOC. Volume 4294 of LNCS, Springer (2006) 327–338

[34] Chris Peltz. Web services orchestration and choreography. IEEE Computer, 36(10):46–52, October 2003.

[35] Hector Garcia-Molina and Kenneth Salem. Sagas. In Umeshwar Dayal and Irving L. Traiger, editors, SIGMOD Conference, pages 249–259. ACM Press, May 27-29 1987.

[36] Jim Gray. The Transaction Concept: Virtues and Limitations (invited paper). In Very Large Data Bases, 7th International Conference, pages 144–154. IEEE Computer Society, 1981.

[37] DAVIES, JR., C. T. 1973. Recovery semantics for a DB/DC system. In Proceedings of the ACM Annual Conference. ACM, 136–141.

[38] RANDELL,B.,LEE,P.,ANDTRELEAVEN, P. C. 1978. Reliability issues in computing system design. ACM Comput. Surv. 10, 123–165.

[39] GARCIA-MOLINA,H.ANDSALEM, K. 1987. Sagas. InProceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). ACM, 249–259.