Toward Undoing in Composite Web Services

Marie-Claude Gaudel

LRI, Paris-Sud University & CNRS, Orsay, France
`mcg@lri.fr`

Abstract. Cancelling or reversing the effect of a former action is a necessity in most interactive systems. The simplest and most frequent form of this facility is the "undo" command that is available in usual, individual, text or graphic editors. As soon as collaborative work is considered, undoing is more intricate since the notion of a last action is not always meaningful. Within this framework, the so-called "selective undo", which allows selecting and cancelling any (or rather some…) former action, has received lot of attention. There are some similarities between cooperative work and composite web services: Component web services are concurrently accessed; they may be treated as shared documents for undoing former actions. Among the latest results on undoing in group editors, the transformational model seems suitable for generalization to other kinds of distributed systems. It completely avoids backward state recovery and allows the selection and cancellation of any former operation. We present some relevant aspects of this model, and then some hints on how to transpose it into the framework of composite web services.

1 Introduction

Dependable composition of web services, and web services architecture, are likely to play a major role in developing the next generation of distributed systems. Reusing solutions from distributed systems techniques seems a natural perspective. However, most solutions will not be reusable directly, mainly because of the openness of the Internet. For instance: every component web service is used and shared by a very large and a priori unknown class of users (persons or other web services); component web services may appear or disappear, etc.

This paper brings in some contribution to the dependable composition of web services, by studying how some aspects of dependability could be addressed in web services architecture. Dependability, in closed distributed systems, often relies on the concept of transaction, which solves both issues of concurrency control and failure occurrences [Gray 1993]. In an open environment the concept of transaction is no more suitable. Transactions may be long lasting, and locking a service for a long time is not acceptable. Another issue is backward recovery: Recovery or cancellation of operation is necessary for every involved component when a composed operation fails for some reason (site crash or user-initiated cancellation). Backward recovery is hardly acceptable in the presence of cooperation-based mechanisms over autonomous component systems such as web services.

One solution to this concern lies in forward recovery: It makes it possible to address dependable service composition in a way that neither undermines the web service's autonomy nor increases their individual access latency [Tartanoglu et al. 2003 a]. It comes to cancelling, or reversing, or compensating the effect of a former action. In interactive systems, the simplest and most frequent form of this facility is the "undo" command that is available in our usual, individual, text or graphic editors. It provides a way of performing the so-called "linear undo": Only the last action is undoable, and then the previous one, and so on; Moreover, there are cases where it is impossible to undo this last action. The implementation of linear undo is based on some history buffer coupled with a "redo stack" (the history buffer behaving as a stack, as well). The "undoing" itself is realized either via state recovery or via the relevant reverse action.

As soon as collaborative work is considered, as it is the case for distributed group editors, linear undo is no more suitable since the notion of a last action is not always meaningful. Within this framework, the so-called "selective undo", which allows selecting and cancelling any (or rather some…) former action, has received a lot of attention [Karsenty and Beaudouin-Lafon 1993, Berlage 1994, Prakash and Knister 1994, Dix et al. 1997, Sun et al. 1998, Ressel and Gunzenhäuser 1999, Sun 2000].

There are some similarities between cooperative work and composite web services: Component web services are concurrently accessed and modified. They may be treated as shared documents when undoing former actions, as discussed above.

Among the latest results on undoing in group editors, the transformational model presented in [Sun 2000] seems suitable for generalization to other kinds of distributed systems since it avoids completely backward state recovery and allows the selection and cancellation of any former operation in the history buffer, under the condition that there exists a reverse operation.

We present some relevant aspects of this model, and then we give some hints on how to transpose it into the framework of composite web services.

2   Doing and Undoing in Collaborative Work

The transformational model considers three meta-commands, namely *do(O)*, *undo(O)* and *redo(O)*, where *O* is an instance of any operation of the collaborative environment. We briefly present the way *do(O)* is dealt with, since it is a necessary introduction to the way *undo(O)* is realised. The way *redo(O)* is performed is not addressed here because currently it does not seem to be of general interest for composite web services.

There is a distinction between the site where an operation *O* is generated and immediately executed, and the other sites where the *do(O)* command is received later on, very likely in a different context (i.e. a different history).

The principle of the transformational model is that operations received from other sites are transformed, according to the local history, before being executed.

The classical example for introducing operation transformation considers the same initial state "abc" for two users [Sun 2000]. User 1 generates *O1 = insert[2,X]*. His or her intention is to insert X between "a" and "bc". Concurrently, user 2 generates *O2 =*

insert[3, Y]. His or her intention is to insert Y between "ab" and "c". The execution of the command *do(O2)* when received by user 1 must transform *O2* into *O'2 = insert[4, Y]*, including in *O2* the impact of *O1*, before executing it.

The management of transformations and executions must ensure the three following properties: convergence, causality preservation and intention preservation.

Convergence. Convergence requires that when the same set of operations has been executed at all sites, all copies of the shared document are identical.

Causal ordering preservation. It is a classical notion in distributed systems. Let operation O_a generated at site i, and operation O_b generated at site j, $O_a \rightarrow O_b$ (read O_b is *causally dependent* on O_a) if and only if:

– $i = j$ and O_a was generated before O_b
– $i \neq j$ and the execution of O_a at site j happened before the generation of O_b

Causality preservation requires that for any dependent pair $O_a \rightarrow O_b$, O_a is executed before O_b on all sites. An operation O_x is said to be *causally ready* at site k if it has been received and not yet executed at this site, and all the operations O_y such that $O_y \rightarrow O_x$ were already executed on this site.

A related relation is that two operations O_a and O_b are said to be independent (noted $O_a \parallel O_b$) if and only if neither $O_a \rightarrow O_b$ nor $O_b \rightarrow O_a$.

Intention preservation. Intention preservation requires that the effect of the execution of *do(O)* in a remote site must achieve the same effect as executing *O* at its original site, at the time of its generation; moreover, the execution effects of independent operations do not interfere.

Convergence and causal ordering preservation are classical properties. They can be ensured by well-known techniques [Lamport 1978, Raynal and Singhal 1996] by associating some vector time stamp to every operation when generated. The innovative and specific notion is intention preservation. It is the key to avoid long lasting blocking, and roll back or backward recovery. Intention preservation is realized by some transformations of the parameters of *O* in order to take into account the difference of context, which is due to the fact that different operations may have been done at the receiving site. A history buffer must be maintained on every site in order to keep the information necessary to the determination of the right transformations when doing a remote operation.

2.1 Transformations

Let us come back to the example above. Operations *O1* and *O2* are independent. The transformation of *O2 = insert[3, Y]* into *O'2 = insert[4, Y]* is achieved by the so-called "inclusion transformation" *IT*: *O'2 = IT(O2, O1)*. In this precise case:

IT(insert[i1, c1], insert[i2, c2]) = insert[i1, c1] when i1 < i2 and insert[i1+1, c1] otherwise.

Remark. Note that this approach has no pretension to solve conflicts. In the example, if i1 = i2 and c1 ≠ c2, there is a conflict that must be detected and solved in some

way. Even if this method can help at detecting conflicts [Molli et al. 2003], solving them is out of its scope.

Conversely, in some cases it is necessary to exclude from the operation O_x to be done the impact of some other operation O_y via the "exclusion transformation" $ET(O_x, O_y)$. It is the case when independent operations are generated from different document states, for instance: Operations O_1 and O_2 are generated independently, respectively by user 1 and user 2, and then, just after O_2, before the propagation of O_1, user 2 generates O_3. When received by user 1, (after O_2, because of causality preservation), the impact of O_2 on the parameters of O_3 must be excluded before including the effect of O_1.

It is worth noting that very often *IT* and *ET* come to identity, as it is the case in the example for couples of *insert* operations.

IT and *ET* are supposed to be defined for every couple of operations of the collaborative system.

2.2 A Generic Schema for Controlling Operation Transformations

The general schema for applying operation transformations is generic. It is independent from the application and the operations. In [Sun et al. 1998], the authors give a general algorithm that can be (rather drastically) summarized as follows:

- When a causally ready operation has its original history being the same as the current history of the receiving site, it can be executed as it is;
- When a causally ready operation has its original history being different from the current history (due to preceding executions of independent operations), this operation needs to be transformed.

To determine the transformation to be applied to a new operation, the algorithm only uses its vector time stamp and the local history. The state vectors time stamping of the operations make it possible to know whether two operations are independent. Moreover, there is in every site a so-called "minimum state vector table" (one state vector by site) that is used to know if an operation is causally ready, to compare its local history with the current history and to perform some kind of garbage collection in the history buffer. This last point avoids keeping the whole history by eliminating operations that cannot be involved in future transformations (see [Sun et al. 1998] for more details and discussion on compatibility with undoing).

2.3 Undoing

Undo is realised on the basis of the transformations above, under the assumption that for any operation O a reverse operation noted $\underline{O}$ is available. Let us consider the case where $Undo(O_i)$ is generated or received at site k, with history buffer $HB_k = O_1...O_iO_{i+1}...O_n$. $Undo(O_i)$ will be performed by the execution of $\underline{O}'_i$ obtained by transformation of $\underline{O}_i$ such that: $O_1...O_iO_{i+1}...O_n\underline{O}'_i$ has the same effect as $O_1...O_i\underline{O}_i O'_{i+1}...O'_n$. Thus there is no rollback.

The transformation of $\underline{O}_i$ into $\underline{O}'_i$ includes the impacts of $O_{i+1}...O_n$, and, in the history buffer, the transformation of $O_{i+1}...O_n$ into $O'_{i+1}...O'_n$ excludes the impact of O_i.

O_i is not kept in the history buffer, but the fact that O_i was done and undone is, for making possible some *Redo(O_i)*.

More precisely, using two straightforward generalizations of *IT* and *ET* to lists *L* of operations whose impacts must be included *(LIT(O,L))* or excluded *(LET(O,L))* before doing operation *O*, the following treatments are performed at run-time:

– $\underline{O}'_i$ is computed and executed, with $\underline{O}'_i = LIT(\underline{O}_i, HB_k[i+1, n])$
– The new history buffer is computed, namely: $O_1...O_i\downarrow O'_{i+1}...O'_n$, with $O'_x = ET(O_x, O_i)$ for $i+1 \leq x \leq n$. O_i is still present, but marked by some symbol (here $\downarrow$) as a "do-undo pair". This allows an efficient treatment of some possible subsequent *Redo(O_i)*.

Note that it is equivalent to doing $\underline{O}_i$, with $O_x \rightarrow \underline{O}_i$ for $1 \leq x \leq i$, and $\underline{O}_i \parallel O_x$ for $i+1 \leq x \leq n$.

This approach has been implemented for various applications where it is necessary to maintain the consistency of shared data: group editors, graphic ones [Sun and Chen 2002] or textual ones [Sun et al. 2004], file synchronizer [Molli et al. 2003]. Here we suggest to use it for slightly different purpose, namely to develop adequate transactional attitudes for web services.

3 Undoing in Composite Web Services via Operational Transformations

In composite Web services, doing and undoing operations of component services is easier because there is no distributed common document to maintain in a consistent state. However, the problem remains that some operations requested by other sources than the composite web service can be performed by the component service. These operations must be taken into account when undoing even the last operation previously requested to a component service.

In the example of Figure 1, a composite Web service, *CWS*, sends operations O^c_1 and then O^c_2 to the *WS* Web service. Between the executions of O^c_1 and O^c_2 by *WS*, *WS* performs operations O^w_1 and then O^w_2 requested by some other sources than *CWS*. Moreover, after O^c_2, it performs O^w_3. Then it receives some Undo command from CWS.

Let us consider the case of *Undo(O^c_2)*. The history buffer of *WS* is $O^c_1 O^w_1 O^w_2 O^c_2 O^w_3$. Following the transformational approach, if we assume that *WS* provide a reverse operation $\underline{O}^c_2$ for O^c_2 and the *IT* and *ET* transformations, the undo command can be realised by

– Execution of *IT($\underline{O}^c_2, O^w_3$)*, including the impact of O^w_3 and
– Transformation of the history buffer into $O^c_1 O^w_1 O^w_2 O^c_2\downarrow ET(O^w_3, O^c_2)$

Similarly, the command *Undo(O^c_1)* from *CWS*, with the same history buffer of WS, is realised by

– Execution of *LIT($\underline{O}^c_1, <O^w_1 O^w_2 O^c_2 O^w_3>$)* and
– Transformation of the history buffer into $O^c_1\downarrow ET(O^w_1, O^c_1) ET(O^w_2, O^c_1) ET(O^c_2, O^c_1) ET(O^w_3, O^c_1)$
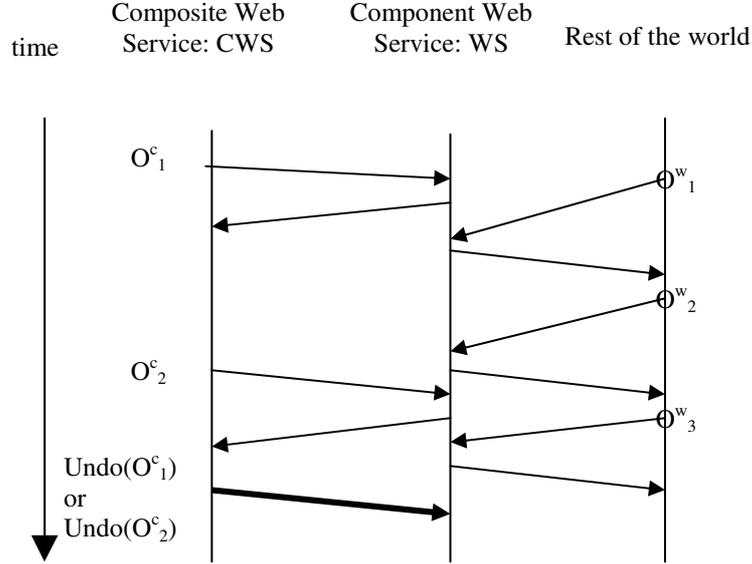
Fig. 1. Example of a scenario when undoing in composite web services

In the general case, let $HB = HB_{pre}.O.HB_{post}$ the history buffer of a web service that receives the *Undo(O)* command, this command is realised by:

– Execution of $LIT(\underline{O}, HB_{post})$
– Transformation of HB into $HB' = HB_{pre}.O\downarrow.LET(O, HB_{post})$

It turns out that in the context of composite web services, the algorithm for applying operation transformations is much simpler than in the case of collaborative work. When some *Undo* command is received by a component web service, only the local state of this web service is concerned. Causal ordering is no more an issue since the only order to be considered is the local total order of execution of the component web service that is registered in the history buffer.

However, the operations of web services are of a different nature than the operations of group editors. It turns out that *IT* and *ET* may require a bit more than modifications of parameters.

3.1 Transformations in the Case of Web Services

Let us consider an example from the classical travel agent case study [Mikalsen et al. 2002, Tartanoglu et al. 2003a]. The travel agent service assists the user in booking complete trips according to his/her requests and is built by composing several existing web services (e.g., accommodation and flight booking, and car rental web services) located through a public web services registry. Each web service is an autonomous component, which is not aware of its participation into a composition process.

We consider one of the airline reservation systems that the travel agent uses.

Let O^c_2 be a successful flight reservation, and O^w_3 another reservation for the same flight. If O^w_3 has been successful, then $IT(\underline{O^c_2}, O^w_3) = \underline{O^c_2}$ and $ET(O^w_3, O^c_2)$ is just O^w_3. If O^w_3 has not been successful, then $IT(\underline{O^c_2}, O^w_3)$ must remove the pending reservation from the waiting list and satisfy it, in order to get a result similar as if O^c_2 was never executed. In this case $ET(O^w_3, O^c_2)$ consists in modifying the indication that O^w_3 was not satisfied. It turns out that the history buffer changes of type of content. It needs to keep more than a simple list of executed operations: for instance, indications on the success or failure of the reservation operations.

The transformations are dependent on the application, but they also depend on an adequate definition of "has the same effect as" in the sentence: $O_1 \dots O_i O_{i+1} \dots O_n \underline{O'}_i$ has the same effect as $O_1 \dots O_i \underline{O}_i O'_{i+1} \dots O'_n$ (cf. Section 2.3). It would be unrealistic to require the system to be exactly in the same state as if O_i never happened. Thus the transformations must be designed in function of some weaker state equivalence: for instance that the same set of travellers has got reservations on the flight.

However, in some cases this is not obtainable. In the example, if the waiting list of the flight was full, O^w_3 will not be satisfied, even if O_2 is undone. Thus some approximation of state equivalence must be considered.

Defining such a notion and the corresponding operation transformations is a major design activity: depending of the underlying application of the web service, the designer must decide what approximation is acceptable and thus, what are the transformations required , and what kind of information must be kept in the history buffer.

But the simplified version of the transformational approach that we propose here provides a framework and some guidelines for this activity. Even more, it gives some possibilities of formal verification [Imine et al. 2003].

IT and ET must satisfy some reversibility properties. In [Sun 200] they are summarised as:

$$O'_a = IT(O_a, O_b) \Rightarrow O_a = ET(O'_a, O_b)$$

It simply means that including the effect of O_b in the parameters of O_a, and then excluding this effect yields O_a again.

In our case, it is possible to weaken this property by taking into account that it is only required in equivalent states. Let us note $\equiv_s$ the relation between states discussed above. Let HB_a (resp. HB_b) the history buffer when Op_a (resp. Op_b) was generated, and $[HB_a]$ and $[HB_b]$ the corresponding states. Then IT and ET must satisfy the following properties:

$$[HB_a] \equiv_s [HB_b] \Rightarrow [HB_a . O_a] \equiv_s [HB_a . ET(IT(O_a, O_b), O_b)]$$

$$[HB_b] \equiv_s [HB_a . O_a] \Rightarrow [HB_a . O_a] \equiv_s [HB_a . IT(ET(O_a, O_b), O_b)]$$

In [Imine et al. 2003] the authors report how they used the SPIKE theorem prover to check these properties on various existing sets of transformation functions, discovering several incorrections or over-specifications. They are currently modifying the SPIKE theorem prover in order to build an integrated development environment for transformation functions. Such an environment would provide a highly valuable assistance for the design of web services with Undo possibilities.

3.2 Transactional Attitudes of Component Web Services

This model provides an elegant solution for undoing in distributed systems without backward recovery. It is applicable under the following conditions:

– For any operation, there is a reverse one
– Every site manages a history buffer
– For every site there are definitions of *IT* and *ET* for all couples of basic operations.

The local algorithm for computing and managing the operation transformations and the history buffer is much simpler than in the case of group editors: there is no problem of causal ordering preservation, since only the local order of execution matters; there is no problem of convergence of different versions of the state of the web services since the only state to be modified is the local one.

This approach is practicable if the vocabulary of operations of each web service is not too large, otherwise the work required for the definition of *IT* and *ET* becomes too important. Besides, there is a significant constraint of efficiency: the web service must be locked during the computation of the transformations of the reverse operation and of the history buffer. Thus these computations must not take too much time. Actually, very often no transformations are needed as it has been observed for group editors [Sun and Chen 2002], [Sun et al. 2004].

For a given web service, the statement of the set of transformations and of the control algorithm corresponds to the definition of a so-called "transactional attitude". Such transactional attitudes are presented as some essential ingredients for the composition of autonomous web services by several authors, for instance [Mikalsen et al. 2002, Pires et al. 2002].

Such a transactional attitude leads to some relaxed notions of transactions based on the "run and then compensate if needed" strategy, where backward recovery is never required. It is compatible with the coordinated forward recovery solution presented in [Tartanoglu et al. 2003 b] for designing dependable composite web services.

It is an interesting alternative to other possible transactional attitudes, such as "pre-commit and wait for a commit or an abort". Both kinds of attitudes could coexist in a composite web service, depending on the context (how frequent are compensations?) and the application behind the component web services (what is easier: to pre-commit or to undo?).

A point in favour of this model is that, as soon as the underlying application of a web service provides reverse operations, it is possible to implement it as a wrapper of the web service [Fabre et al. 2003]. The role of the wrapper is to receive the requested operations, to transform them when needed, and to manage the history buffer. An example of the adaptation of a single-user editor (namely MS Word) to the operational transformation technique is reported in [Xia et al. 2004].

4   Conclusion and Perspectives

In this article, we show how to transpose techniques developed for collaborative work and group editors to composite web services. The so-called Operational Trans-

formation approach provides an elegant general model for undoing operations in multi-user applications avoiding backward recovery.

This approach can be simplified and adapted into an "optimistic" transactional attitude: we call it optimistic because it is based on a "run and then compensate if needed" strategy. It is an interesting alternative to other more classical transactional attitudes, such as "pre-commit and wait for commit or abort". As soon as the underlying application of the web site provides reverse operations, this transactional attitude can be implemented without modifying it, as a front-end or a wrapper.

It is possible to formalise this approach and to formally verify the correction of the transformations, thus of the compensation strategy.

This model and the corresponding transactional attitude seem worth to explore.

Acknowledgement

References

[Berlage 1994] Berlage, Th.: A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects. ACM Trans. on Computer-Human Interaction, 1(3), 269-294, 1994

[Dix et al. 1997] Dix, A., Mancini, R., Levialdi, S.: The Cube Extending Systems for Undo. DSVIS'97, Granada 1997

[Fabre et al. 2003] Fabre, J-C., de Lemos, R., Gacek, C., Gaudel, M-C., Georgantas, N., Issarny, V., Levy, N., Romanovsky, A., Tartanoglu, F.: Final Results on Architectures and Dependable Mechanisms for Dependable SoSs. Deliverable DSC1 of the DsoS IST Project, April 2003, http://www.newcastle.research.ec.org/dsos/

[Gray 1993] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993

[Imine et al. 2002] Imine, A., Molli, P., Oster, G., Rusinowitch, M.: Development of Transformation Functions Assisted by Theorem Prover. 4th International Workshop on Collaborative Editing (ACM CSCW 2002), New Orleans, Louisiana, USA, November 2002

[Imine et al. 2003] Imine, A., Molli, P., Oster, G., Rusinowitch, M.: Proving Correctness of Transformation Functions in Real-time Groupware. Proceedings of the 8th European Conference on Computer Supported Cooperative Work, 14-18 September 2003, Helsinki, Finland, pages 277-293, Kluwer, 2003

[Karsenty and Beaudouin-Lafon 1993] Karsenty, A., Beaudouin-Lafon, M.: An Algorithm for Distributed Groupware Applications. IEEE ICDS'93 Conference, 193-202, 1993

[Mikalsen et al. 2002] Mikalsen, T., Tai, S., Rouvellou, I.: Transactional Attitudes: Reliable Composition of Autonomous Web Services, DSN 2002 Workshop on Dependable Middleware-based Systems (WDMS 2002), 2002

[Molli et al. 2003] Molli, P., Oster, G., Skaf-Molli, H., Imine, A.: Using the Transformational Approach to Build a Safe and Generic Data Synchronizer. Proceedings of GROUP 2003, ACM 2003 International Conference on Supporting Group Work, November 9-12, 2003, Sanibel Island, Florida, USA. ACM, 2003

[Pires et al. 2002] Pires, P. F., Benevides, M. R. F., Mattoso, M.: Building Reliable Web Services Compositions. International Workshop on Web Services Research, Standardization, and Deployment - WS-RSD'02, 551-562, 2002

[Prakash and Knister 1994] Prakash, A., Knisler, M.: A Framework for Undoing Actions in Collaborative Systems. ACM Trans. on Computer-Human Interaction, 4(1), 295-315, 1994

[Raynal and Singhal 1996] Raynal, M., Singhal, M.: Logical Time: Capturing Causality in Distributed Systems. IEEE Computer Magazine 29(2), 49-56, 1996

[Ressel and Gunzenhäuser 1999] Ressel, M., Gunzenhäuser, R.: Reducing the Problem of Group Undo. ACM conf. on Supporting Group Work, 131-139, 1999

[Sun 2000] Sun, C.: Undo any Operation at any time in Group Editors, Proc. of ACM conf. on Computer-Supported Cooperative Work, 2000

[Sun et al. 1998] Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving Convergence, Causality-preservation, and Intention-preservation in Real-time Cooperative Editing Systems. ACM Trans. on Computer-Human Interaction, 5(1), 63-108, 1998

[Sun and Chen 2002] Sun, C., Chen, D.: Consistency maintenance in real-time collaborative graphics editing systems. ACM Transactions on Computer-Human Interaction, 9(1), 1-41, 2002

[Sun et al. 2004] Sun, D., Xia, S., Sun, C., Chen, D.: Operational transformation for collaborative word processing. Proceedings of ACM 2004 Conference on Computer Supported Cooperative Work, Nov 6-10, Chicago, 2004

[Tartanoglu et al. 2003 a] F. Tartanoglu, V. Issarny, A. Romanovsky, N. Lévy : Dependability in the Web Service Architecture, Proc. of ICSE 2002 Workshop on Architecting Dependable Systems, Orlando, Florida. LNCS 2677, 89-108, Springer-Verlag, 2003

[Tartanoglu et al. 2003 b] F. Tartanoglu, V. Issarny, A. Romanovsky, N. Lévy : Coordinated Forward Recovery for Composite Web Services, 22nd IEEE Symposium on Reliable Distributed Systems, Florence, 2003

[Xia et al. 2004] Xia S., Sun D, Sun C., Chen D., Shen H. : Leveraging Single-user Applications for Muti-user Collaboration : the CoWord Approach, CSCW'04, 162-171, November 6-10, 2004