INTEROPERABLE TRANSACTIONS FOR E-BUSINESS

Sergei Artishchev, Hans Weigand

S.Artyshchev@uvt.nl, H.Weigand@uvt.nl
Infolab, Tilburg University
The Netherlands

Abstract: For enterprise systems to interoperate, as in an e-business transaction, some kind of transaction management is crucial. Current standard transaction protocols do not meet important requirements for e-business transactions. In this paper, a new transaction model is introduced that meets those requirements and for which well-defined business semantics can be given. *Copyright © 2005 IFAC*

Keywords: transactions, logic

1. INTRODUCTION

Interoperability has been defined as "the ability for a system or a product to work with other systems or products without special effort of the part of the user", or alternatively as "the ability of Enterprise Software and Applications to interact". Increasingly, enterprises are cooperating with other enterprises. Not only large organizations set up cooperation agreements with other enterprises, but also SMEs are combining forces to compete jointly in the market. Nowadays, an enterprise's competitiveness is largely determined by its ability to seamlessly interoperate with others. A crucial interoperability element at the application level is the management of interoperable transactions.

Initially transactions were developed for database functionality. Further research advances made them applicable to other architectures, e.g. workflows, messaging systems, etc. With the emergence of Internet and, more importantly, its becoming an e-business infrastructure, transaction concepts were applied to Web Services architectures. To achieve such flexibility a concept underwent several rounds or evolution, each of them brought new models developed and applications implemented.

To execute successfully, transactions utilize various mechanisms one of which has a resource reservation application – locking. Locking is used extensively in so-called blocking protocols - protocols that require exclusive access to the resource for completion. Despite early E-Business protocols attempts to avoid locking for an extensive time period, locking as a form of resource reservation can and should still be applied as a mechanism to establish a mutual commitment leading to contract fulfillment. Together with other non-obligatory preparatory mechanisms, locking precedes transaction execution and is part of a pre-transaction phase.

Database locks are applied for the whole duration of transaction. *Advanced transaction models* (ATM) represent distributed business processes. The following characteristics differentiate them from simpler transaction models thus leading to increased complexity:

a) Business processes are composed activities – topologically distributed, arranged as multilevel nested trees, and addressing various processes with their own internal functionality characteristics. This complexity creates issues of correctness criteria selection (traditional ACID is relaxed) and how to preserve them using locking or other mechanisms;

b) Execution duration is time-extended (transactions are long-running activities), creating a requirement to minimize lock duration and/or use alternative correctness-preservation mechanisms;
c) As ATM execution addresses distributed business entities, locking gets a different semantics. A lock not only affects the access status of the resource, but also imposes an obligation on a locked participant to function in a predefined expected way;
d) The changes caused by the execution of ATM activities might not be reversible to the exact previous state

Those characteristics were addressed by ATM (Saga, ACTA, etc.) with the following solutions:
a) Realizing that intermediate states of participants could be observable by other participants, Atomicity and Isolation criteria are relaxed: resources' locking for the whole duration of global transaction isn't performed and intermediate results of execution steps might be visible.
b) Because strict locking is not applicable, two approaches were developed: 1) a global transaction is divided into fine-grained sub-transactions (if their local atomicity can be preserved) in such a way that locking at this level might be possible; or 2) transactions are realized to be non-atomic and executed without locking of their participants. Rather, compensations (as introduced in SAGA) are used to undo results of completed execution units if the global transaction is aborted.

Other ATMs (flexible transactions (Elmagarmid, 1995), interoperable transactions (Weigand and Ngu, 1998)) brought a notion of alternative execution paths and preferences, attempting to minimize the quantity of compensations, for performance reasons.

In this paper, we first summarize the requirements on interoperable transactions posed by e-business applications, and discuss briefly the most important current standards. Then we introduce our e-business transaction model and discuss how it meets the requirements. Section 3 describes the basic semantics of the locking in Deontic Dynamic Logic.

2. E-BUSINESS TRANSACTIONS

E-business transaction is a persistent change of participants' state aimed to achieve a certain (predefined) business goal using Internet-based architectures and technologies (Lewis, et al., 2001; Papazoglou, 2002; Tygar, 1998). E-business transactions ultimately address goods/services vs. payment exchange (transactions in the economic sense) and their execution is typically governed by some legal framework.

2.1 E-business transaction requirements

The following requirements of e-business transactions distinguish them from ATMs:
- Participants are dynamic (*volatile*);
- while initially advertising their functionality in UDDI-type registry, they might change as a whole or some of its characteristics only;
- therefore locks should address actual functionality, not assumed or initial;
- Participants are *autonomous* entities, sometimes executing opportunistic behavior. Locks could be removed not only by Requestor, but also by Provider itself;
- The transactions' technical infrastructure (media, protocols, etc.) might be *unreliable*, creating issues of recovery and preservation of locks;
- Rather then being focused on execution speed, e-business transactions use schedules and timeouts (specified duration); being contract-governed *execution duration* is predefined. While almost insignificant for traditional transaction models, process execution isn't instantaneous and is subject to coordination and scheduling. Minimizing execution time of certain steps is a responsibility of the Provider;
- Participants have *functional* and *capacity restrictions* on their operations. Functional restrictions reflect internal business capabilities of the participants, which are hard to change, capacity restrictions address participants' characteristics at the execution time.

The principal difference between database and e-business transactions is based on the resource behavior. For a database transaction, a resource is an entry in a database system (passive entity) and the owner is reactive only. For business transactions the resources are dynamic and owners are active agents that can decide whether and how they will execute a certain request. In complex scenarios, there might be even a negotiation about the behavior of the locked resource (e.g. rules on unlock and self-unlock) before lock application is attempted.

2.2 Current standards

The most significant recent developments on e-business transactions include BTP, BPEL, and WS-CAF that we discuss briefly.
BTP (BTP, 2002) – represents e-business execution as a nested transaction governed by two-phase outcome protocol. The protocol performs outcome determination; each sub-transaction has provisional and final effects. Lock-alike behavior is achieved through confirmations of effects. However, achieving of provisional effect already requires performing of some activity, and, in case it can't complete successfully, compensation is necessary. All reservations are performed as part of the execution: a notion of pre-transaction resource capability verification is absent due to an optimistic bias towards participants' functionality.

BPEL (**WS-T** (WST, 2002)) - has two major advantages: extensibility and a well-developed flow control language. Transaction aspects are addressed in WS-T as part of the protocol, separately covering atomic transactions and business agreements. Transaction properties of the former are ACID while the emphasis of business agreements is on outcome determination avoiding the notion of locks.

WS-CAF (**WS-TXM** (WSTXM, 2003)) is a recently proposed specification addressing transactions designed specifically for web services. While a notion of preparation is introduced as a mechanism to determine participants' readiness, it isn't explicitly bound to context-derived obligations. The notion of locking is vague due to the fact that the protocol draws on context-based coordination and outcome determination as operational mechanisms.

Roughly speaking, we can say that the current standards do support typical advanced transaction solutions, in a flexible way, but do not meet important requirements imposed by e-business transactions. Sometimes this is an intentional choice. According to (Little and Freund, 2003), letting business logic affect the transaction flow "blurs the distinction between what you would expect from a transaction protocol (guarantees of consistency, isolation) which are essentially non-functional aspects of a business "transaction", with the functional aspects". Apart from the fact that the "expected" (consistency, isolation) is precisely *not* guaranteed in the relaxed variants of WS-TXM, the consequence of this choice is that the e-business transaction logic is now to be hidden in the application code or process models, which is not desirable either. We would say that the business logic as such (the costs of self-unlock, how to compensate etc) is something that is neither in the transaction model itself, nor in the application but is specified in between (the contract, cf. (Weigand and Van den Heuvel, 2002)) – but the transaction layer can contribute by providing well-defined primitives for locking, preparing, etc. This is worked out in the next section.

2.3 E-business transaction model

In order to cope with the requirements above, e-business transactions should be modified compared to those of ATM. Transaction, following common business rationale, is divided into three phases (Figure 1). Each phase includes certain common business steps that lead to achievement of a business goal.. Issues of transaction goal determination, establishing business criteria, composition, etc. precede transaction execution and are beyond the scope of the proposed model. Similarly, performance analysis, functionality adjustment etc. are also not a part of a transaction.

The first phase is a transaction execution preparation or **pre-transaction**. It is split into two sub-phases: *prepare* and *locking*. Prepare is a verification (confirmation) of prospective participants' functionality while locking indicates agreement of a Web Service to participate is a transaction. Technically, both prepare (check) and e-business locking are (a)synchronous message exchanges between prospective participants and a change of Provider's state (and a context). Each step is followed by verification – checking if a transaction can be executed with available participants functionality. A phase (or each step) could be followed by saving context, which could be retrieved later or even used for another transaction.

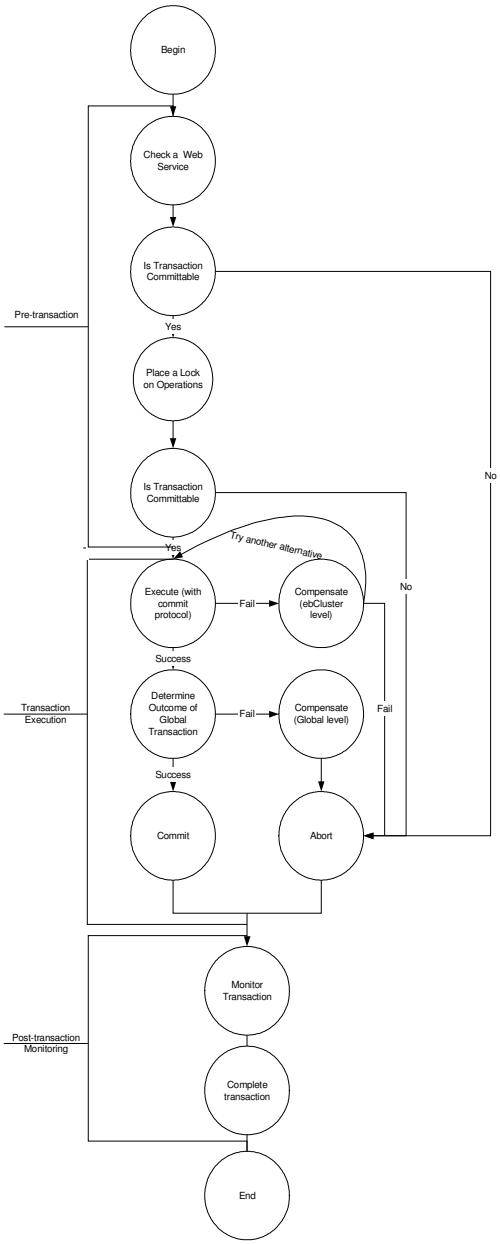

Fig.1.Transaction execution flow.

Execution is a phase addressed by other execution protocols and designated to execute transaction prepared at previous step. Execution itself might be single protocol-based (like BTP) or flexible (WS-T), where several protocols could be utilized. Only one execution protocol is invoked, it must be supported by all participants. In a proposed protocol execution protocol is negotiated at pre-transaction phase between prospective participants. An outcome of each execution step is determined upon its

completion and, in case of failure, a (local) compensation is executed. Unlike in other protocols, an outcome determination is also included into the execution phase, because, if execution outcome is determined to be negative, a global compensation might be executed. Each participant is informed about outcome and unlocks its resources. Upon execution completion of main transaction results (context) are saved again.

The final protocol phase is **post-transaction monitoring**. The phase is included because even thought global transaction execution might succeed, the whole transaction still could fail due to un-fulfillment of post-transactional obligations which are results of transaction execution (e.g. warranties, maintenance, etc.). In essence it is a monitoring of contract clauses against failure for an extended (pre-defined) period of time. Global transaction is completed only when post-transaction monitoring is completed successfully.

The proposed model provides the following advantages:

- the proposed protocol is contract-governed, while for other protocols business logic is implemented in the execution flow;
- separation into phases to better meet business and technical issues and that could be addressed by different protocols thus enabling flexibility;
- checking and locking are mechanisms to ensure correct execution through establishing obligation-based relationships between participants at pre-transaction phase, eliminating uncertainty during execution (risk management) and eventually minimizing transaction cost (due to possible decrease in sub-transaction failures);
- post-transaction monitoring allows fulfillment of extended post transaction obligations without it being part of the main transaction, thus enabling externalization of the transaction's results as soon as they could be determined and used.

Although the distinction between a prepare and a locking step is made in several e-business transaction models, the exact meaning of "preparing" and "locking" is not always clear. For example, in their discussion of WS-Transaction, Alonso et al (2003) state that for web services it is "difficult to characterize the notion of resources, locks and rollback" (p.226). In this paper, we suggest the following characterization:

Prepare phase – at this phase we propose *functionality verification of prospective participants*. Being part of the pre-transaction, it precedes both locking and execution. Other preparatory mechanisms (protocol-specific) might involve application of holds, initialisations, soft locks, etc. We assume this phase's activities verify functionality of prospective participants and don't impose any definite reservation, neither they form an obligation (therefore, the terms lock and hold are misleading). In case of completion or execution failure the preparation doesn't require compensation.

Because participant's profile, initially published in a registry might reflect actual functionality incorrectly (being outdated) or incompletely (containing insufficient information to invoke provider's functionality), the requesting party might want to check this information. Checking (or verification) is performed either at the provider itself (if it is trusted party) or at a third party.

While functionality restrictions are quite rigid, *capacity restrictions* vary due to resources' utilization by other parties. Exact capacity value (or, rather, available capacity) is correct only at lock application, however, any estimate performed before invocation also contributes to the efficiency of the execution scenario because it allows excluding (potentially) unavailable participants from the scenario without the need of a lock.

Querying a registry is an optional part of the pre-transaction phase not to be confused with checking, typically preceding it. To perform a query a Requestor should know the location of the required registry and the API to manipulate it (or have its access brokered) and should have rights to access it. A Registry itself is of UDDI, ebXML or any other standard (supported type), is itself an independent entity, and is known before the transaction begins.

Locking Phase – some protocols employ explicit locking to ensure prospective participants' enlistment. A lock creates *obligations*. This phase follows the prepare phase and assumes the existence of a contract or similar agreement specifying the type of locks to be applied and their characteristics. The lock type should be supported both by requesting party and by resource owners.

The model can be further refined by noticing that the preparing applies either to a participant or the participant functionality. The same is true for the locking (Fig. 2).

	Participant	**Functionality**
Prepare	**Prepare Participant**	**Prepare Functionality**
Lock	**Lock Participant**	**Lock Functionality**

Fig.2. Phase/object orthogonal representation

Participant lock serves as a gateway to functionality (or in case of web-services, operation) locks. It allows the use of functionality of the locked participant limited by arrangements specified at the time of locking. The basic semantics is that of authorization: it provides the Requestor with a right or capability to lock an operation, or, equivalently, it obliges the Provider to grant locking requests (under certain arrangements). This type of lock might be the only one needed if the participant provides only one operation or if it can receive all necessary information to apply operations locks transitively.

Functionality (operations) lock is an actual locking, it creates mutual obligation between participants. This lock is used as an execution correctness preservation mechanism. Given the participants' autonomy, it might be up to the provider to designate an actual operation to be executed, depending on request characteristics.

This orthogonal architecture provides the following benefits:

Firstly, it allows transitivity of properties and functionality from Provider to Resources it controls, thus optimising *speed* of locking (no need to provide additional information for every operation);
Secondly, *execution costs* of a transaction are minimized due to the following optimizations: early actual functionality detection and late locking. Early actual functionality detection (registry querying and preparation) allows to decide if a participant could be enlisted for a transaction and to establish a correct contract, but also allows the locking to be postponed as long as possible. This is an important advantage over models in which these functions are combined in one operation.
Both prepare and locking phase precede transaction execution and should not be included in the execution phase, as (Little, 2003) seems to do, but their impact is quite different. While prepare verifies functionality and requests additional information, lock is applied upon known functionality; in the case of locking, compensations might follow for unlocking, while prepare is a request for information with no compensations defined or needed. The provider is considered to be a *prospective* one before lock application and *actual* after.

3. LOCKING SEMANTICS

In this section, we provide formal semantics for the locking and unlocking operations in terms of Dynamic Deontic Logic (Wieringa et al, 1989). This logic allows for the specification of *actions* (locking, unlocking, requesting, compensating etc) and for the specification of *obligations* (which we need to model the commitment aspect of e-business locks). We start with some general "common-sense" semantics of locking and requesting.
Let L be a first-order language. DDL is L extended with deontic operators (see below) and a dynamic operator. That is, if φ is a wff in DDL and α is an *action*, then [α]φ is a wff in DDL, with the intuitive meaning: φ holds after action α has been performed. The action can also be composite: $\alpha_1;\alpha_2$ stands for sequential, and $\alpha_1 \&\& \alpha_2$ stands for parallel execution. ¬α stands for not-doing α.
In this case, we have at least the action *lock(r,id,t)* and *unlock(id),* where *r* is a resource, *id* a lock identifier and *t* a lock type (worked out below). The following minimal axioms hold:

Definition 1 *(general semantics of lock and unlock)*

$$\forall r,id,t\ [lock(r,id,t)]\ lock\text{-}ed(r,id,t)$$
$$\forall r,id,t\ lock\text{-}ed(r,id,t) \Rightarrow [\neg unlock(id)]\ \neg lock\text{-}ed(r,id,t)$$

These axioms just state that a resource is locked by a lock action, and remains locked until an unlock action is performed. We identify a lock by some unique identifier rather than by the resource as sometimes multiple locks on the same resource are allowed. Note that for simplicity we omit here any variable typing.
The lock operation is performed by the *Provider* (this agent is not included as a parameter, as we deal here only with one Provider at a time). If a *Requestor* wants a lock, he has to send a request message to the Provider. The Provider either accepts or rejects this request (negotiations are not in the scope of this paper). In the case of an accept, the Provider gets an obligation to perform the lock. In fact, this communication logic holds not only for lock but also for unlock and any other operation that the Provider could perform.

Definition 2 *(semantics of request)*

$$\forall R,P,m,\alpha\ [request(R,P,\alpha,m);\ accept(P,R,m)]\ Obl(P,R,\alpha)$$
$$\forall R,P,m,\alpha\ \neg Obl(P,R,\alpha) \Rightarrow [request(R,P,\alpha,m);\ reject(P,R,m)]\ \neg Obl(P,R,\alpha)$$

In this case, the *m* acts as identifier of the request. Obl is the deontic operator for obligation. The most important property of Obl is expressed in the following axiom:

$$Obl(P,R,\alpha) \Leftrightarrow [\neg\alpha]\ Violated$$

which says that if P is obliged to R to perform some action, not doing that action leads to a *violation*.

The operational semantics of locking are explored here by considering the intended lock properties one by one. *Exclusiveness* indicates if a locked resource is available for utilization by several parties. If exclusive lock is applied, other attempts to apply exclusive lock are denied.

Definition 3a *(exclusive lock excludes other locks)*

$$\forall r,id,id2,t\ lock\text{-}ed(r,id,exclusive) \Rightarrow [lock(r,id2,t)]\ false$$

Saying that an action leads to false is equivalent to saying that the action is not possible in the modal sense. So this axiom says that a resource locked exclusively, cannot be locked again (whatever the lock type). What is even more important is that it also cannot be *used*. In our formal model, this can be expressed in two ways: either by defining *r* to be impossible if it is done for any other requestor, or by defining it forbidden for the Provider to grant requests for others. We choose the second solution that has the advantage that it fully preserves the autonomy of the Provider as far as the execution of the service is concerned.
We still should be able to distinguish legitimate requestors (the one that set the exclusive lock) and others. We can solve this by assuming that the lock id is a kind of ticket, or secret key, and the request is

not legitimate when the requestor does not connect this id to the request.

Definition 3b *(exclusive lock excludes providing the service)*

```
∀r,id,P,R,m lock-ed(r,id,exclusive) ⇒
 [request(R,P,do(r),m) && ¬ pass(R,P,id)]
       For(P,accept(P,R,m))
```

Definitions 3 incorporate the semantics of "exclusive", but they do not say anything yet about what happens when the legitimate requestor appears, with the right *id*. Does it mean that in that case the *P* is *obliged* to accept? That would mean that *R* would be able to charge *P* if for some reason, *P* does not grant the request (perhaps because the resource is no longer available). If *P* wants a firm commitment from *R* that the resource is and remains available for him, this is something that is independent from the exclusiveness property. For this purpose, we introduce the notion of reserve lock as a conditional obligation.

Definition 4 *(reserve lock)*

```
∀r,id,P,R,m lock-ed(r,id,reserve) ⇒
 [request(R,P,r,m) && pass(R,P,id)]
       Obl(P,accept(P,R,m))
∀r,id,P,R,m lock-ed(r,id,reserve) ⇒
 [request(R,P,r,m); do(r)]
       Perm(P,unlock(id))
```

The first rule says that *P* becomes obliged to accept the service request. *P* could still for some reason fail to accept the request, but then this leads to a violation, and makes him liable for sanctions or compensations to be specified separately.
The second rule says that when the Provider has performed the requested operation, he is permitted to unlock and thereby lift the obligation. This is the normal situation: the obligation is fulfilled when the service has been performed. However, the abnormal situations are also relevant and need to be specified. For reasons of space, the logic of these specifications is not included in this paper.

3. CONCLUSION

The proposed protocol addresses e-business transactions from an integrated technical and business perspective. Business semantics are important and need to be explicated. Unlike other e-business protocols our protocol separates the transaction into three major phases thus enabling utilization and combination of various mechanism at each phase. The notions of checking (prepare) and locking operations are introduced explicitly for the first time as well as requirements operations are provided. None of the current protocols support those mechanisms in the scope defined. Furthermore contract-originated (obligation-based) locking is a new concept that addresses reliability of execution environment by excluding potentially unreliable participants at an early (pre-transaction) phase. None of the current web service standards supports post-transaction monitoring. The proposed architecture addresses it by introducing a separate phase with provision to use various monitoring protocols.

Acknowledgement
This work is partially supported by the Commission of the European Communities under the 6th framework programme (INTEROP Network of Excellence, Contract N° 508011). See http://www.interop-noe.org .

REFERENCES

Alonso, G., Casati, F., Kuno, H., Machiraju, V. (2004). *Web services – Concepts, Architectures and Applications*. Springer Verlag.

Lewis, P.M., Bernstein, A., Kifer, M.(2001). *Databases and Transaction Processing: An Application Oriented Approach*, Addison-Wesley.

Elmagarmid, A. (Editor) (1995).*Database transaction models for advanced applications*, Morgan Kaufmann, San Mateo.

Litte, M. (2003). Transactions and Web Services. CACM, **46(10)**: pp.9-54.

Little, M., Th.J.Freund (2003). A comparison of web services transaction protocols – a comparative analysis of WS-C/WS-Tx and OASIS BTP. Dowloaded 31-08-04 from http://www-128.ibm.com/developerworks/webservices/library/ws-comproto/

Papazoglou M.P.(2002). The world of e-business : web services, workflows, and business transactions. Notes from Keynote Speech, at CAISE2002, Toronto, May 2002.

Tygar, J.D.(1998). Atomicity in Electronic Commerce, ACM-Mixed Media.

Weigand, H., A.H.H. Ngu, (1998). Flexible Specification of Interoperable Transactions. *Data & Knowledge Engineering*. 25(3): pp.327-345.

Weigand, H., W.J.A.M. van den Heuvel (2002). Cross-organizational workflow integration using contracts. *Decision Support Systems*, 33(3), 247-265.

Wieringa, R.J., J.-J.Ch.Meyer, and H. Weigand, "Specifying dynamic and deontic integrity constraints", *Data & Knowledge Engineering* 4(2), 1989, pp.157-191.

Business Transaction Protocol (BTP). An OASIS Committee specification V.1.0 June 2002; http://www.oasis-open.org/committees/download.php/1184/20020603.BTP_cttee_spec_1.0.pdf

Web Services Transactions (WS-T). Joint spec by IBM, Microsoft, and BEA, August 2002; http://www-106.ibm.com/developerworks/library/ws-transpec/

Web Service Transaction Management (WS-TXM). Joint Specification by Arjuna, Fujitsu, IONA, Oracle, and Sun V1.0 July 2003 http://developers.sun.com/techtopics/webservices/wscaf/wstxm.pdf