CONTRACT-BASED INTEROPERABILITY FOR E-BUSINESS TRANSACTIONS

Sergei Artyshchev, Hans Weigand

Infolab, Tilburg University, PO Box 90153,
5000 LE Tilburg, The Netherlands
sergei@uvt.nl, h.weigand@uvt.nl

Abstract. Due to the heterogeneous nature of web services, interoperability is a crucial aspect, and this interoperability has not only a data aspect (ontologies) but also a process aspect To ensure correctness of e-business processes, some kind of transaction support (e.g. WS-TXM [16]) is necessary but not sufficient. In this paper, we define correctness in terms of contract compliance. Being contract-guided, e-business processes require a contract-dependent transaction protocol that can be represented as a set of formally specified agreements and obligations between participants. In current standards and frameworks, contract-based interoperability is only referred to, but not specified in details. This paper contributes to contract-based interoperability is the following aspects: for the first time it attempts to classify characteristics for various degrees (levels) of contract-based interoperability; it provides formal deontic logic definitions of basic support operations - check and lock -, and their semantics is defined.

Keywords: e-business, transactions, deontic logic, locking, interoperability

1. Introduction

Interoperability of web services is a crucial aspect of the enterprise integration. Interoperability can be loosely defined as the ability of enterprise software and applications to interact. True interoperability is more than connectivity and communication. It includes the possibility that one role performs some activity for the other role, and so it assumes that there is shared understanding of what the meaning of the request is: both the content semantics (activity name, parameters) and the pragmatics (the intended effect, e.g. that the other role executes the request or sends a reject message). This "shared understanding" can be implicit in the code, or be more explicit in an agreed-upon protocol definition, "collaboration agreement" (ebXML), or "contract". In this paper, we are interested in contract-based interoperability, defined as: "the ability of applications to interact and work together on the basis of a contract", where a contract is defined as: "an agreement between two or more roles

that governs their interaction in terms of obligations and permissions". A contract need not be explicit, although this does have certain advantages. In principle, every interaction is contract-based, as every interaction assumes certain semantics/ pragmatics of the communication to be in place, but typically these pragmatics are implicit in the standard protocol that is imposed from the beginning. Several frameworks (ebXML, WS-Coordination) provide the participants with the possibility to define new or extended protocols (agreements) with specifically defined semantics. When we use the term "contract" in this paper, we mainly refer to these user-defined agreements, but sometimes we also use it in the more general sense of the agreement underlying any interaction.

We will define different levels of contract-based interoperability, depending on the level of support of various contract manipulation procedures. Common procedures for contract manipulation include but are not limited to: contract establishing, contract verification, contract evolution during transaction execution, contract monitoring, sub-contract handling and inclusion, etc.

This paper is organized as follows: first we introduce the contract-based interoperability levels; then, section 3 introduces basic concepts to achieve contract-based interoperability, such as locking; section 4 provides operational semantics for these basic concepts, and the final section contains conclusions and future work guidelines.

2. Levels of Contract-Based Interoperability

In the context of collaborative business development, contract-based interoperability of web services can be divided into six categories. Both external and internal contract-related functionalities are the basis for classification. Each higher-level category includes functionality of lower level.

2.1 Contract-based interoperability levels

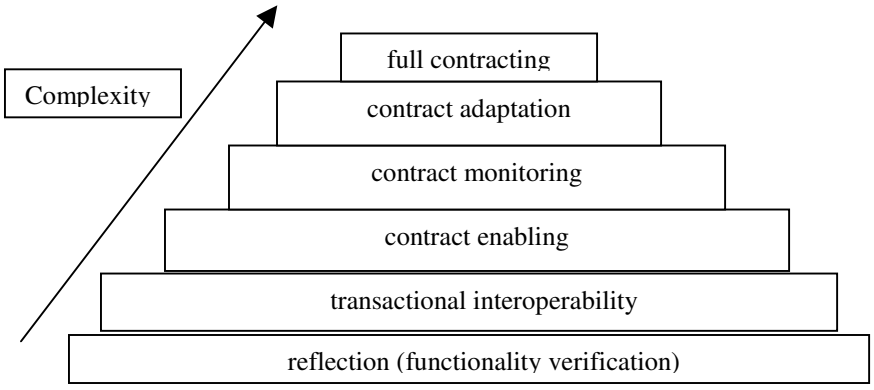

Fig. 1. Six levels of contract-based interoperability

Level 0 indicates that no contract-based interoperability features are supported. Internal functionality of participants of this level allows execution of pre-designated operations, however, only final results (and, sometimes, intermediate status) are externalised. Other participants could locate its profile in the registry, however, no communication except for request for service/result output, is possible. It can execute advertised functionality, but no interoperation protocol support is defined.

Level 1. A participant can not only **advertise,** but also **confirm (verify)**, if requested, its functionality and make a choice for the most appropriate operation (of the same type) to be employed at a run-time. Internally a participant should have a request processor which performs communication with other parties and, if necessary, a self-test mechanism that can generate latest status/performance relevant characteristics upon request. Externally, such participant allows request processing (with appropriate authentication, if necessary), basic two-way communication, and a check operation (to be defined in section 3).

Level 2. A participant supports one or (preferably) several transactional protocols. Support of level 2 interoperability indicates **transactional interoperability** – the capability to be engaged in transactions with some kind of (relaxed) ACID properties. To correspond to this level, a web service should support at least one transaction protocol and this capability should be advertised in a registry.

Level 3. The basic functions are in place that allow participants to make commitments and fulfill them. This assumes message-processing capabilities (to accept, process and respond to incoming messages), transactional interoperability, but, most importantly, what is supported at this level is outcome reservation (locking, to be defined in section 3), outcome determination, and other basic **contract-enabling** operations.

Level 4. Participants can **monitor a contract**. Monitoring contract assumes that participants are capable to understand, execute and verify compliance of other parties' activities to contract clauses. *Understanding* the contract means the capability to interpret contractual clauses (expressed in some XML-based contract definition language), and support the operations defined in the contract. *Execution* refers to the internal functionality to fulfill obligations assumed as part of a contract. Finally, *monitoring* itself refers to the capability to verify other parties' activities against contract clauses and response with contract-defined corrective actions. Contract monitoring has been the subject of several recent research projects [6,10,12].

Level 5. Participants cannot only execute a given contract, but also **adapt a contract** by means of negotiation and refinement. This level requires rather developed conversation capabilities and support of obligation-based contract composition. At this level participants are not yet assumed to establish a complete contract from scratch, rather, they should reuse already existing contacts (or templates), compose a contract from other contracts (as in supply chain scenario) or refine already existing

contract with clauses and parameters relevant to the concrete business scenario. This level has been explored in [11] in the context of agent societies.

Level 6 Participants are able not only to refine contract templates, but also to **setup a new contract,** typically on the basis of explicit *goals* and preference structures from each participant. This functionality implies the use of goal-based negotiations and the ability to extract or compose contractual clauses from sources other then pre-defined templates and samples.

Those proposed interoperability levels could be utilized to characterize the level or degree to which a participant is interoperable as part of the architecture. If an architecture as a whole is specified in terms of proposed levels, then the interoperability specifications for each participant might be derived. Another possible implementation might be to use these levels in the WSDL description in a registry thus facilitating the discovery of business partners with appropriate characteristics.

Contract-based interoperability can also be used to assess current state-of-the-art technology. If we look at current web service standards such as BTP [14] and WS-Transaction [15], we can characterize them as level2. They do provide transactional interoperability by means of which parties can synchronize a certain event, but the business semantics of these synchronized events (in some cases called "BusinessAgreement") are not specified. Therefore, there is also no monitoring of obligations, the only monitoring, if any, concerns the transaction protocol execution. If we look at current agent models (mainly confined to research labs) dealing with contracts (see the references above), we can characterize them as level 4 to 5. Level6 technology doesn't exist yet. However, contract drafting is a research topic in the area of negotiation support and e-commerce, see e.g. [7]. If we look at the ebXML framework, we can observe that in principle, it supports all levels of contract interoperability. However, in practice collaboration profile agreements (CPA) are still composed manually, and the notion of commitment or obligation is not explicit, so ebXML is better characterized as level2.

The objective of this paper is to close the gap between web service technology and agent theory by addressing precisely the level that is still not supported by current web service standards and, at the same time, is largely implicit in the agent models – that is, the basic level3 contract functions. We do not have a complete list yet of what these basic functions are, but we claim that the check and lock operations introduced in section 3 are very essential. The formal semantics are given in section 4.

3. Locking as an Interoperability Mechanism

In the previous section we defined locking as a level3 contract-enabling function. In this section, we list the main requirements on e-business transactions [13] and then introduce our locking model. For a more elaborate discussion of the model, see [2].

3.1 Locking Requirements

The following requirements of e-business transactions distinguish them from advanced transaction models (ATM) defined in the '90s [3]:

- 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 exhibiting 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.
- Participants have functional and capacity restrictions on their operations. Functional reflect internal business capabilities of the participants that are hard to change, capacity addresses participants' characteristics at the execution time.

Although their semantics have shifted since the time they were introduced in the context of database transactions, *locks* are still a valuable mechanism to arrange mutual reservation on participant's capacity. E-business locking guarantees that either a functionality is exercised as requested or the party, incurred losses due to another party' obligation un-fulfillment, is compensated, as it is agreed upon when lock is applied (or specified in contract).

The principal difference between database and e-business transactions is based on participants' behavior. A database is a passive entity and the owner is reactive only. In business transactions the participants 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 behavior of locked resource before lock's application is attempted.

3.2 E-business locking model

In order to cope with the requirements listed in section 3.1, e-business locks should be modified compared to those of ATM. Transaction preparation might be split into two sub-phases: prepare and locking [4]. Technically, e-business locking is an asynchronous message exchange between prospective participants and a change of the Provider's state in case lock is applied successfully. While not being explicitly addressed, e-business transaction execution is enclosed with support/service phases addressing reliability and correctness issues.

Although the distinction between a prepare and a locking phase is made in several e-business transaction models, the exact meaning is not always clear [1]. We suggest the following characterization:

Prepare phase – at this phase we propose *functionality verification of prospective participants*. Being part of pre-transaction phase of typical e-business transaction, it precedes both locking and execution. Other preparatory mechanisms (protocol-specific) might involve application of holds, initializations, soft locks, etc. We assume this phase's activities verify functionality and capacity of prospective participants and don't impose any definite reservation, neither they impose an obligation on any party (therefore, the terms lock and hold may be misleading). In case of completion or execution failure the preparation doesn't require compensation. Because the 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 by the provider itself (if it is trusted party) or by 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 efficiency of execution scenario because it allows excluding (potentially) unavailable participants from the scenario without the need of a lock.

Locking Phase – some protocols employ explicit locking to ensure prospective participants' enlistment. A lock creates *commitments*. This phase follows the prepare phase and typically, it assumes the existence of a contract or similar agreement specifying the type of locks to be applied and their characteristics, but this contract can also be more general and implicit. The lock type should be supported both by requesting party and by functionality-proving participants.

As we said, checking applies either to a participant or the participant capacity (its available resources). The same is true for the locking (Table 1).

	PARTICIPANT	**CAPACITY**
PREPARE	**Check Participant**	**Check Capacity**
LOCK	**Lock Participant**	**Lock Capacity**

Table 1. Phase/object orthogonal representation

Participant lock serves as a gateway to capacity (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. 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. This kind of lock is not exclusive.

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

This orthogonal architecture provides the following benefits:
- it allows transitivity of properties and functionality from participant (Provider) to capacity (resources) it controls, thus optimizing speed of locking (no need to provide additional information for every operation);
- it minimizes cost and quantity of compensation. Application of participant/capacity locks could be relatively extended in time, allowing reservation costs to be minimized. For example, a check can be performed in January, a participant lock in March (for the rest of the year), a capacity lock in June, when the production planning is finalized, while the actual execution is only performed in October. When the check in January fails, another supplier can be looked for. Similarly, when the participant lock fails in March, another pre-selected supplier can be chosen. Etc. In this way, the risks are minimized against minimal costs.

Both prepare and locking phase are preliminary to transaction execution, but their impact is quite different. While check verifies functionality and requests additional information, lock is applied upon known functionality; the check request is based on *advertised* functionality, while lock is based on *confirmed* (verified) functionality; in the case of locking, compensations might follow for unlocking, while check 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.

4. A formal model of e-business locking

In this section, we provide formal semantics for the locking and unlocking operations in terms of Dynamic Deontic Logic [8,9]. 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).

4.1 Basic Definitions

Let L be a first-order language. DDL [8,9] 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 $[\alpha]\varphi$ 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. $\neg\alpha$ 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)*

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

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 P 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*. Note that this scheme is a bit naive about time; normally, some deadline will be specified, and the violation only arises when the action is not done before the deadline [5]. For the time being, our abstraction suffices. The semantics of request applied to locking results in:

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

that is, the Provider is obliged to lock the resource. If this does not result in a locking action (which can never be excluded, given the autonomy of P), then P certainly has something to explain (within a certain marketplace or agent society, P may get trouble with the market owner).

4.2 Operational semantics of locking (basic model)

The operational semantics of locking are explored here by considering the intended lock properties one by one. Because of space limitations, we omit our definition of exclusive lock and define reserve lock only. Suppose that an "exclusive lock" has been set, then no one else can use the resource, but what happens when the legitimate requestor appears, with the right *id*. Does it mean that in that case the Provider is *obliged* to accept the request? We do not think this is necessary in all cases. That would mean that *P* has reserved *r* for *R*, and 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 at all). If *R* wants a firm commitment from *P* 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 (see below). Note that we take *reserve* and *exclusive* to be orthogonal properties: although they will often go together, other combinations are also possible. For example, reserve but not exclusive: for the requestor in a e-business transaction, usually the most important thing is that he can use the resource, and putting an exclusive lock would be only a means to achieve that. The combination exclusive/no reserve may be useful for example in maintenance situations where some party wants to prevent the resource to be used for some time but without the intention to use it himself. The combination non-exclusive/no reserve is possible theoretically, but then there are no specified effects of the locking at all, and so the operation looses its meaning.

The second rule says that when the Provider has performed the requested operation, he is permitted[1] 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. What about the Provider unlocking the resource on his own initiative? In a business context, this may very well occur, for various reasons. For example, because the production capacity of the Provider went down, or because a better-paying service request comes in from another Requestor. Because this may very well occur, it is customary to specify some compensation when it happens, either in the contract or together with the locking request.

If the Provider removes the reserve lock, this leads automatically to the lifting of the obligation (assuming a closed interpretation of the rule in definition 4), so we do not specify that explicitly. However, we do specify that self-unlock is forbidden, that is, leads to a violation. The consequences of that violation differ from one contract to another and can't be specified here.

Definition 5 (self unlock is forbidden, but not impossible)

```
∀P,id ¬Perm(P,unlock(id)) ⇔ For(P,unlock(id))
```

[1] Permitted is a strong version of the deontic P operator. It implies that doing the action does not lead to a violation (in other words, is not forbidden), but is stronger because it must be set explicitly, cf. the concept of authorization [8].

So unlocking is forbidden unless permitted (which is at least the case after successful performance, def. 4). We might want to specify in addition that self-unlock, if it happens, leads to an obligation to inform the requestor about this unhappy failure. This would be a good rule for a *locking message protocol*, but such a protocol is beyond the scope of this paper.

Intention and commitment

Usually, a lock also expresses an intention of the requestor to use the resource. The Provider can see the lock operation as a sign that the resource is going to be used, and could anticipate on that. However, from the agent literature we know that the semantics of "intentions" is rather weak, so we refrain from formalizing it.

What can be important in e-business transactions is that the Requestor commits himself to use the resource. A commitment is much stronger than an expression of intent. It means that the requestor is liable to charges (compensation) if he doesn't use the resource. This can be very relevant in business transactions: the Provider may loose interesting business operations because of the locking, and so the lock has an economic value. One could think of various ways to formalize this commitment. The following rule is again naive about timing, but suffices for the idea, Applying a reserve lock creates a commitment to use the service (unless unlocked later).

Definition 6a *(reserve locking creates a commitment)*

```
∀R,P,r,id,m
[request(R,P,lock(r,id,reserve),m);accept(P,R,m)]
       locked(r,id,reserve) ⇒ Obl(request(R,P,r,_))
```

We must also account for the situation that the Requestor requests an unlock. In that case, he authorizes the Provider to unlock. Usually, when there is a penalty on not using the service, there is also a penalty for the Requestor on unlocking (otherwise, the first penalty could be easily circumvented), but the conditions and details often differ. For example, the conditions may say that unlocking is possible without penalty till a certain point in time. The following definition deals with unlocking by the Requestor. It says that the Provider should always grant an unlock request (he can not ignore it if that would be more convenient), but at the same time, it is regarded as something the Requestor should not do (that is, it is forbidden and liable to penalty).

Definition 6a *(unlocking is forbidden, but not impossible)*

```
∀R,P,id,m [request(R,P,unlock(id),m] Obl(P,accept(P,R,m))
∀R,P,r,id locked(r,id,reserve) ⇒
              For(R,request(R,P,unlock(id),_))
```

Compensation

In traditional transaction models (database and advanced) rollback brings participants to the state preceding the locking attempt, while in the case of failure of an e-business

transactions return to the previous state might be unnecessary or impossible. Rather, compensation is used to bring the participants into some state specified by a contract. In the formal model developed in this paper, compensation can be defined as an action specified in the contract that removes a violation of one of the obligations or prohibitions around locking. Note that in the above, we already have identified a number of possible violations, such as failure to deliver the locked service, failure to use the locked service and pre-emptive unlocking. Each violation requires a different compensation. The details of these are specified in the contract, or agreed upon at the time of the lock request, or may even follow from general law.

Definition 7 *(violations can be compensated)*

$$\forall X,Y,\gamma,r,n\ (\mathrm{violated}(X,r,n) \wedge (\mathrm{violated}(x,r) \Rightarrow \mathrm{Obl}(X,\gamma)) \wedge \neg\mathrm{compensated}(X,Y,r,n)) \Leftrightarrow [\gamma]\ \mathrm{compensated}(X,Y,r,n)$$

Here X and Y are participants (subjects), r is some rule identifier, n is used to identify a particular violation of that rule (the rule might be violated several times), and γ is the compensatory action specified in the contract. We assume here that the DDL is powerful enough to distinguish different violation predicates and is able to count their instances. Note that, strictly speaking, compensation does not remove the violation (which remains as an historical fact, so to say), but only specifies when it is compensated, which is sufficient. Note also that the compensatory action is itself an obligation, and when it fails it may necessitate additional compensations.

Participant lock

Up till now we have considered locking of resources (capacity) only. What is a reasonable semantics for participant locking? We propose to define participant locking as an authorization by means of which P commits himself to accept locking requests (perhaps conditionally, only when capacity is available at that time):

$$\forall R,P,m,\alpha\ [\mathrm{request}(R,P,\mathrm{lock},m)]\ \mathrm{Obl}(P,R,\mathrm{accept}(P,R,m))$$

This helps to increase the chance that the business transaction will be successful. It is not needed in all circumstances, but in some uncertain situations, it can be helpful.

5. Conclusions and Future Work

While technical aspects of transactions are currently addressed by several industrial standards and protocols, many contract business-related issues are left at the discretion and implementation of business partners, in many cases leaving interoperability issues open. Therefore, a general contract-based interoperability definition should be provided and mechanisms for its contractual specifications should be defined. This paper addresses transactions from contract-based prospective introducing five levels of interoperability and providing descriptions for their

characteristics. The main focus is on contract enabling, which is implemented as a set of e-business transaction steps: check and lock.

Further development of contract-based interoperability involves adaptation of one of the available conversation languages to contract negotiation. There is also a need for more empirical research on the business requirements for web service technology.

Acknowledgements
This work has been partially supported by NWO (Netherlands Organisation for Scientific Research) and the European NoE INTEROP.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: *Web services – Concepts, Architectures and **Applications***. Springer Verlag, 2004.
2. Artyshchev, S, Weigand, H. "Interoperable transactions for e-business". Submitted.
3. Elmagarmid, A. (Editor) "*Database transaction models for advanced applications*" Morgan Kaufmann, San Mateo, 1995.
4. Little, M. "Transactions and Web Services" *CACM*, 46(10): pp.9-54, 2003.
5. Marjanovic, O. Milosevic, Z.. "Figaro Should Be in Sydney by the 2nd of July" - Contracting in Many-To-Many e-Services. B. Schmid et al (Eds.): *Towards The E-Society: E-Commerce, E-Business, and E-Government*, (Proc. I3E 2001). Kluwer 2001, pp.431-444.
6. Z. Milosevic, G. Dromey, "On expressing and monitoring behaviour in contracts" In: *Proc. 6th Int. Conf. on Enterprise Object Computing (EDOC'02)*, Lausanne, Switzerland, September 2002.
7. Y.H. Tan, W. Thoen,"Electronic Contract Drafting Based on Risk and Trust Assessment". *Int. Journal of Electronic Commerce 7(4)*, 2003, pp.55-72.
8. Weigand. H., F. Dignum, E.Verharen, 1997. "Integrated Semantics for Information and Communication Systems". In: R. Meersman, L. Mark (eds), *Database Application Semantics*. Chapman & Hall
9. 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.
10. Weigand, H. and Heuvel, W.J.A.M. van den. Cross-organizational workflow integration using contracts. *Decision Support Systems*, 33(3), 2002, pp.247-265.
11. Weigand, H., Dignum, V., Meyer, J.J.C. & Dignum, F. "Specification by refinement and agreement: Designing agent interactions using landmarks and contracts". In Petta, P., Tolksdorf, R. & Zambonelli, F. (Ed.), *Engineering Societies in the Agents World III (ESAW 2002).* (LNCS, 2577, pp. 257-269). Berlin: Springer-Verlag.
12. Lai Xu and Manfred A. Jeusfeld, "Pro-active Monitoring of Electronic Contracts", In. *Proc.15th Conference On Advanced Information Systems Engineering (CAiSE 2003),* 2003. Lecture Notes of Computer Science, Springer-Verlag.
13. J. Yang and M.P. Papazoglou, "Interoperation Support for Electronic Business", *CACM*, Vol. 43, No. 6, pp. 39-47, June 2000.
14. Business Transaction Protocol (BTP). An OASIS Committee specification V.1.0 June 2002; See http://www.oasis-open.org
15. Web Services Transactions(WS-T). Joint specification by IBM, Microsoft, and BEA, August 2002; http://www-106.ibm.com/developerworks/library/ws-transpec/
16. 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