

# An Agreement Centric Access Control Mechanism for Business to Business E-Commerce \*

Victoria Ungureanu †  
Department of MSIS  
Rutgers University  
ungurean@rbs.rutgers.edu

## ABSTRACT

We argue that matrix-based models are inadequate for regulating business to business (or B2B, for short) e-commerce due to the diversity, complexity and potential large number of commercial agreements that have to be supported. To deal with these issues, we propose in this paper an agreement-centric access control model. The paper introduces the concept of communication agreement (*CAR*) as a means for specifying contractual terms, and presents the *CAR* enforcement mechanism. We explore the expressive power of the model and show that it can implement regulations which cannot be expressed using conventional mechanisms alone. The paper also describes a prototype implementation; the preliminary performance results indicate that the enforcement mechanism is quite affordable, even in its present, experimental stage.

## 1. INTRODUCTION

In order to save money, be competitive and be efficient, more and more enterprises are taking steps towards conducting transactions with trading partners on-line [5]. Among the problems inherent in such projects none is more serious than the difficulty to control the activities of the disparate agents involved in e-commerce.

Trading relations between enterprises are based on mutually agreed upon contracts. Generally, these contracts enumerate agents authorized to participate in transactions, and spell out such things like rights and obligations of each partner and terms and conditions of the trade. *Control occurs then as an ancillary to such commercial agreements.*

---

\*Appeared in the Proc. of the 16th ACM Symposium on Applied Computing; Special Track on Web and E-Business Applications, March 2002.

†Work supported in part by DIMACS under contract STC-91-19999 ITECC, and by Information Technology and Electronic Commerce Clinic, Rutgers University, and by Reasearch Council, Rutgers University

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright ACM 0-89791-88-6/97/05 ..\$5.00

Since the seminal work of Lampson [12] access control has traditionally been seen in terms of an access matrix. The access matrix is conceptually simple and the majority of control mechanisms are based on it [10]. However, we will argue that matrix-based models are not fully adequate in B2B context due to the *diversity*, *complexity* and *large number* of commercial agreements that have to be supported. We will now elaborate on these factors and draw conclusions on which this work is based.

First, the access matrix model is not expressible enough to support many policies deemed useful for today's enterprise [9; 3]. Whenever one needs to implement an agreement which does not fall into the supported patterns, there is no other way, but to build from scratch a specific interface for it [3]. But, embedding the terms of the contract into code is time consuming and expensive to carry out. Moreover, hard-coding the contract means that changes are difficult to carry out, often requiring rewriting of interfaces.

Second, it is not uncommon for enterprises to have a large number of of supplier enterprises, each operating under a different contract. For example, Ford has approximately thirty thousand suppliers, and General Motors has about forty thousand [6] (both companies have recently announced their intention to perform their inter-enterprise business online). The current prevailing method for establishing a set of policies is to *combine* them into a single super-policy. But compiling all contracts an enterprise is bound by into an access matrix is a daunting task.

Thus, in B2B context, a huge, ever-changing matrix, subsuming all contracts by which an enterprise is bound, becomes prohibitively hard and error-prone to maintain. Moreover, the matrix is not conceptually needed since a request is regulated solely by a single agreement which can be uniquely identified. To deal with these problems, we propose in this paper an *agreement-centric* access control model. At the foundation of this model is the concept of communication agreement, for short *CAR*. *CARs* are *autonomous* entities embodying contractual terms. In order to make changes easier to perform *CAR* terms are expressed explicitly, in a formal, rule-based language. And, in order to increase the expressive power of the formalism, the conditions and provisions of a contract can take into account its current state.

Communication agreements are brought to bear by generic servers, called *CAR* observers (*CARO*), which can evaluate and carry out the terms of any well-formed *CAR*. To deal with the scalability issue, observers are not a-priori dedicated to any agreement; the terms of a *CAR* are *dynamically*

loaded into an observer<sup>1</sup> only when a request regulated by the *CAR* in question occurs. And once the request is evaluated, a *CARO* may serve any other incoming request, regardless of the agreement under which it was issued. This feature effectively decouples the number of agreements of the number of servers needed to enforce these agreements. Thus, it is possible that a large number of agreements be enforced by a considerably smaller number of servers. The rest of the paper is organized as follows: We start, in Section 2, with a description of the concept of communication agreement; the *CAR* enforcement mechanism is presented in Section 3. We follow, in Section 4 by discussing the expressive power of the formalism and by presenting the concrete implementation of a contract which cannot be expressed using conventional mechanisms alone. Section 5 describes the system implementation and Section 6 discusses related work. We conclude in Section 7.

## 2. THE CONCEPT OF COMMUNICATION AGREEMENT

Agreements/contracts represent the prevalent way for expressing expected rules of conduct of a group of people involved in a certain (economic) activity. The nature of commercial agreements can be illustrated by the following example:

*Agents in a client enterprise, CE, are allowed to purchase from a supplier enterprise, SE, if the following conditions are met:*

- *only agents duly certified as purchase officers by `clientAuthority`<sup>2</sup>, a designated certification authority of CE, may issue purchase orders.*
- *the supplier enterprise honors purchase orders (POs) issued by CE, for up to a certain cumulative value—to be called “blanket”.*
- *only agents duly certified as sale representative by `supplierAuthority`, a designated certification authority of SE, are authorized to respond to POs.*

Let us summarize here a few points that are important for understanding what agreements are at the societal level:

- Agreements are *global* in the sense that they refer to everybody involved in a certain activity; they set common expectations about the behavior of agents subject to them. However, the globality of an agreement does not imply the identity of behavior of agents subject to it. Participants are not equal under an agreement; an agreement distinguishes between agents subject to qualification—e.g. only sales representatives are allowed to respond to purchase orders, past actions—e.g. an order is to be processed only if the value of

<sup>1</sup>An observer is chosen to handle a particular request based on its current load, and on whether it had previously evaluated a request sent under the same contract.

<sup>2</sup>For the sake of convenience, *CARs* provide means for mapping public keys with names. In this example, `clientAuthority` stands for the public key of the authority trusted to sign purchase officer certificates on behalf of the client enterprise.

the purchase does not exceed the current value of the blanket.

- Agreements are *specific* in the sense that they regulate a well defined set of actions, deemed important for the activity in question. For example, the blanket agreement above regulates only the release of POs and their responses and does not refer to other additional activities the agents in question might be involved in.

In the following we will formally define our concept of agreement, which we call *communication agreement (CAR)* in order to distinguish it from the general term. Our goals are to mirror as closely as possible the social notion of agreement, while tailoring it for the specific needs of electronic commerce. We assume that in distributed computing the observable behavior of an agent consists of the messages it sends to and receives from other agents. As such, a *CAR* regulates communication acts between agents.

Specifically, a communication agreement *A* consists of the following elements: (1) the set of *messages* which may be issued under this *CAR*; (2) the *state of A*; and (3) an explicit set of *rules* embodying the contractual terms. We will further elaborate on the components of a *CAR*; to illustrate the structure of a *CAR* and the nature of its rules we will use the blanket agreement (*BA*, for short) introduced previously.

### Messages

A communication agreement, *A*, specifies the type of messages that might be issued under *A* and the format of these messages. In addition, message types can be annotated to specify the evaluation context, i.e. whether state information is required for sanctioning a particular message instance.

In the current implementation, messages are written in XML, and the format of the messages is given using document type definition (DTD) [19] or a Schema [15]. For example, the blanket agreement introduces two types of messages—`purchaseOrder*` and `responseToOrder`—and records DTDs for both of them. The annotation of `purchaseOrder` type denotes that state information—in this case, the current value of the blanket—is required for authorizing a message of this type.

### State

The state maintains relevant information that allows distinctions to be made between different phases of an agreement. For example, the state of the blanket agreement, *BA*, consists of a term `blanket(b)`, where *b* represents the current amount left in the blanket. In the current implementation the state of a *CAR* is represented as a list of Prolog terms.

### Rules

The rules of a *CAR A*, specify the set of conditions under which a request is to be sanctioned by *A*. These conditions may refer to the action thought, the content of the message, the certificates presented and the state of the *CAR*. In addition, a rule may call for updates of the state of the *CAR* in question.

For now, we have chosen to use Datalog [17], a subset of Prolog, due to its expressive power, well-understood seman-

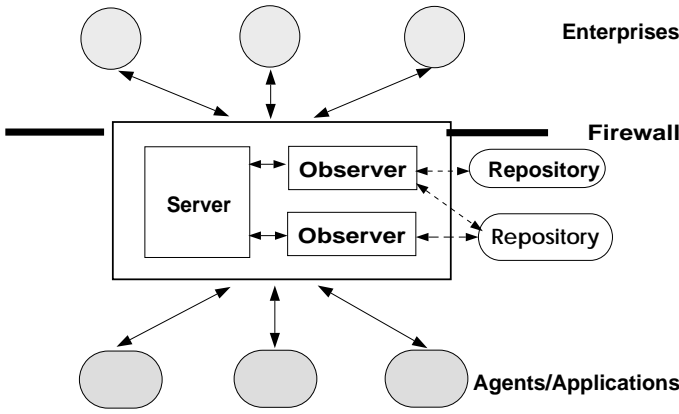


Figure 1: System architecture

tics and its relatively good performance cost<sup>3</sup>. We will show in Section 4 how the terms of the blanket agreement can be implemented as *CAR*-rules.

### 3. THE CARENFORCEMENT MECHANISM

The system architecture, illustrated in Figure 1, relies on the existence of two trusted entities: *CAR* repositories and *CAR* engines. A repository provides persistent storage for rules, states, authorized issuers, message and certificate format of the *CARs* under its jurisdiction. A given agreement  $A$  is maintained by a single repository, called the base of  $A$ . The uniqueness of the base of  $A$  is necessary in order to preserve the consistency of its state, as we shall see later. *CAR* engines, called observers, are generic tools that can verify the format of messages and certificates, and interpret and carry out any well formed set of rules. Under this scheme, each server processing B2B messages has at least one associated observer, to which it passes received requests for evaluation. Servers are trusted to service only requests sanctioned by observers.

We are in position now to explain how the exchange of messages gets to be mediated by observers, and how the enforcement is carried out. Consider a request  $r$  to perform operation  $op$  on a message  $m$ , is issued under an agreement  $A$ . Assume further that  $r$  arrives at a server having an associated observer  $O$ . Then the following steps will be taken:

- Observer  $O$  determines the identifier of the agreement  $A$ , and the address of  $B_A$ , the base of the agreement under which the message was sent. It is worth mentioning that in the current implementation, we use as communication protocol extended HTTP<sup>4</sup> which allows us to include the address of  $B_A$ , in the message header.
- If  $O$  does not have the agreement  $A$ , it will ask  $B_A$  for it. When the repository responds,  $O$  will check whether  $B_A$  is a valid repository, i.e. it is established as such by a trusted organization.

<sup>3</sup>A question is solved in linear time with respect to the number of rules of the Datalog program.

<sup>4</sup>HTTP Extension Framework [8] provides a mechanism for adding additional header and verb information to HTTP requests, such that the nature of the information being conveyed can be inferred without having to delve into the request itself.

$typeOf(M,T)$	binds $T$ to the type of the message $M$ ;
$valueOf(M,T,V)$	succeeds if tag $T$ is present in message $M$ , and binds $V$ to its content;
$replace(T,T')$	replace term $T$ with term $T'$ in the state of the <i>CAR</i> ;

Figure 2: Additional predicates

- If the request is accompanied by certificates then, each certificate is verified, and translated into an internal format—essentially a Prolog list.
- Observer  $O$  determines the type of the message  $m$ , and verifies whether  $m$  is well-formed, i.e. it complies with the format maintained by  $A$ . Furthermore, if  $m$ 's type is annotated,  $O$  will ask  $B_A$  for the state of  $A$ .
- Finally, the observer checks whether there is a rule in  $A$  authorizing the request. If there is, the processing cycle of  $m$  continues; otherwise, the request is discarded.

#### Deployment of Communication Agreements

If there is an organization  $T$ , trusted by all parties involved in an agreement  $A$ , then the observers and the base of  $A$  can be maintained by  $T$ . If this is the case, an enterprise making business transactions under  $A$  can trust that terms of  $A$  are observed by all participating agents, regardless of the enterprise they belong to.

If there is no such trusted intermediary, any enterprise interested in enforcing the terms of an agreement can establish its own observers and bases. (This deployment scheme is depicted in Figure 1). Given that mutual trust between B2B participants is not usually assumed, if the latter deployment scheme is employed then an enterprise cannot rely on the other participants to observe an agreement. It follows, then that if an agreement  $A$  is say, between a pair of enterprises  $E$  and  $E'$ , then *both* of them need to enforce  $A$ , which leads to some duplication of effort.

### 4. CAR RULES FORMULATION

The rules of a communication agreement,  $A$ , state the conditions that have to be satisfied in order for a request to comply with  $A$ , and the effect, if any, the request should have. Rules are expressed as Datalog predicates of the form:

$authorized(Op,M,C)$ ,

where  $Op$  is the requested operation,  $M$  is a handle to the message to which  $Op$  refers to, and  $C$  is the evaluation context. The evaluation context is comprised by the list of certificates presented by the user translated into an internal format. And if the processed message is annotated, then the evaluation context also contains the state of *CAR*  $A$ . In addition to the predefined Datalog predicates, the body of a rule may contain some additional predicates, presented in Figure 3.

The rest of the section is organized as follows. We start by presenting how the terms of the blanket agreement ( $BA$ )

introduced in Section 2 can be implemented as *CAR*-rules; and we follow by discussing the range of agreements that can be implemented as *CARs*.

#### 4.1 An Example: The BA Rules

The terms of the blanket agreement can be implemented by the four rules presented in Figure 3. Before we proceed with the rule description, few comments about the format of messages accepted by this agreement. As mentioned previously, BA regulates two types of messages<sup>5</sup>: purchase orders, which are defined as such by a term `type(purchaseOffer)` and replies to POs, which are recognized by a term `type(responseToOrder)`. The value of an offer is given by the value of the `amount` tag.

Rules  $\mathcal{R}1$  and  $\mathcal{R}2$  regulate the sending, and, respectively, the retrieving of POs as follows. By Rule  $\mathcal{R}1$ , a request to put a PO is authorized only if the following two conditions are met. First, that there is a certificate, `C` in `LC`, the list of certificates presented by the requester, which is issued by `clientAuthority` and certifies the bearer to be a `purchaseOfficer`. Second, that `X`, the value of the purchase, is smaller than `B`, the current value of the blanket. If this is the case, the value of the blanket is decreased accordingly. By Rule  $\mathcal{R}2$ , such a purchase order can be retrieved only by an agent presenting a certificate issued by `supplierAuthority` which establishes him as a `saleRepresentative`.

Similarly, Rules  $\mathcal{R}3$  and  $\mathcal{R}4$  regulate the sending and the retrieving of responses to POs. First, Rule  $\mathcal{R}3$  ensures that only an agent presenting a certificate issued by `supplierAuthority` authorizing him to act as a `saleRepresentative` may respond to a PO. And only an agent duly certified by `clientAuthority` as a `purchaseOfficer` may retrieve such a response (Rule  $\mathcal{R}4$ ).

Note that under these rules any other certificates issued by these, or other certifying authorities, have no established semantics and consequently give no leverage to the bearer.

#### 4.2 Expressive Power

In the following we will compare the expressive power of *CARs* with one of the access-matrix implementation, namely access control lists (ACLs), and will argue that the former is more expressive than the latter.

An ACL-based access control mechanism operates, essentially, as follows [16]: ACLs associate each protected object (or types of objects) with a list of users. For each user  $u$ <sup>6</sup>, a complete description of which operations  $u$  may undertake is given. An access control list for an object  $o$ , consists then of pairs of the form  $\langle u, l \rangle$ , where  $l$  is the list of operations that  $u$  can perform on  $o$ .

Such a mechanism can be implemented using *CARs* as follows: the set of message types recognized by the agreement in question is, in this context, the set of protected objects; the state is empty, since ACLs do not refer to state information; and the set of rules is obtained by having for each pair  $\langle u, l \rangle$ , associated to a message type, a rule of the form:

```
authorized(A,m,LC):- member(C,LC), member(u,C),
member(A,l),
```

<sup>5</sup>This is only a finger exercise, meant to illustrate the mechanism; a full agreement should consider other types of messages, including delivery notices, acknowledgments, etc.

<sup>6</sup>where  $u$  is a name or role derived from a certificate

```

R1. authorized(put,M,[LC,[blanket(B)]]):-
    typeOf(M,purchaseOffer),
    member(C,LC),
    member(issuer(clientAuthority),C),
    member(role(purchaseOfficer),C),
    valueOf(amount,M,X),B>X, B1=B-X,
    replace(blanket(B),blanket(B1)).

R2. authorized(get,M,[LC,[blanket(B)]]):-
    typeOf(M,purchaseOffer),
    member(C,LC),
    member(issuer(supplierAuthority),C),
    member(role(saleRepresentative),C).

R3. authorized(put,M,LC):-
    typeOf(M,responseToOrder),
    member(C,LC),
    member(issuer(supplierAuthority),C),
    member(role(saleRepresentative),C).

R4. authorized(get,M,LC):-
    typeOf(M,responseToOrder),
    member(C,LC),
    member(issuer(clientAuthority),C),
    member(role(purchaseOfficer),C).

```

Figure 3: The Rules of the Blanket Agreement (BA)

where  $A$  is the requested operation,  $m$  is a handle to the request, and `LC` is the list of certificates presented by  $u$ .

So, any agreement that can be implemented using ACLs can also be implemented under *CARs*. The opposite is not true: an ACL based mechanism cannot make use of content request, nor of state information. For example, the requirement of BA-agreement, stating that the value of a purchase order should not exceed the blanket, cannot be expressed using ACLs alone.

## 5. ON THE IMPLEMENTATION OF THE MECHANISM

In the current implementation we are using the Jigsaw server [4], developed by W3 Consortium, which has been modified to communicate with observers. Any server may have one or more observers associated with it; the server chooses one of its associated observer and passes requests to it for disposition.

Observers are implemented mostly in Java, and are generic *CAR* engines in the sense that they can enforce the terms of any well formed *CAR*. An observer  $O$  operates as an independent process and it communicates by pipes with the server it is associated with. Observers receive the body of a *CAR*  $A$ , from its base,  $B_A$ , via authenticated communication.

The main task of repositories are to provide persistent storage for *CARs* and to distribute them to valid observers. Note that *CAR* rules as well as message and certificate description may be replicated on any number of observers because of their static nature. However, a *CAR* has a mutable component: the state. In the following we will discuss the issues raised by *CAR*-state, and the protocol devised to ensure its consistency.

*Preserving CAR-State Consistency*

To increase the expressive power of the formalism, rules can refer to, and also modify the *CAR*-state. And to solve the scalability problem, the terms of a contract may be evaluated by any observer. This increased flexibility comes with the price of having to maintain the *CAR*-state consistent. Two cases have to be taken into account when dealing with the *CAR*-state consistency: (1) requests are processed by the same observer; and (2) requests are processed by different observers. To ensure state consistency when the same observer,  $O$ , evaluates requests issued under the same agreement  $A$ , it is sufficient that  $O$  evaluates the rules pertaining to  $A$  *sequentially*, and carries them out *atomically*, so that the sequence of operations that constitutes the ruling for one request does not interleave with those of any other request sent under the same agreement.

To deal with the case where several observers process requests issued under the same agreement, we have devised a protocol for *CAR* state distribution. The protocol assumes that a repository maintains for each agreement  $A$  under its jurisdiction the following information:  $H_A$ , the last observer that have received  $S_A$ , the state of  $A$ ; and  $L_A$  the list of observers which requested  $S_A$  and are not served yet.

The distribution protocol, illustrated in Figure 4, proceeds essentially as follows. When an observer  $O$  needs  $S_A$  it first checks to see if it has a valid copy. If yes, it proceeds with the evaluation without having to contact  $B_A$ , the base of  $A$ . If not,  $O$  sends a message `getState(A)` to  $B_A$ , to request the state of  $A$ . There are several cases to consider:

- If this is the first request for  $S_A$ , then  $B_A$  sends  $S_A$  to  $O$  without further delay, and records  $O$  as being the current holder.
- If  $S_A$  is held by an observer  $O'$ , then the repository sends to  $O'$  an `invalidate(A)` message. The effect of this message is twofold:  $O'$  invalidates its own copy and sends to the repository the most recent  $S_A$ .
- If the repository is in the process of serving another request for  $S_A$ , then it appends  $O$  to its list of unserved requests. Otherwise, the repository sends the state of  $A$  to  $O$ , via a message `state(S(A))`.

Once  $O$  receives the requested state it computes and carries the ruling for the  $A$ -request in question. And as long as its own local copy of  $S_A$  is valid, any other  $A$ -request assigned to  $O$  for disposition can be evaluated without requiring the extra communication with the base of  $A$ . It follows then that overall performance is improved if previous assignments are taken into consideration when distributing tasks to individual observers.

The protocol achieves *sequential consistency* [11], i.e. it satisfies the following condition: the messages received under an agreement  $A$  are processed in some arbitrary order (strict chronological order is not required), but *all observers perceive the same order*, which is imposed by the base of  $A$ .

## 6. RELATED WORK

There has been a growing interest in e-commerce, and in the *specification* of the agreements regulating this type of commerce [2; 7]. But specification of a contract is only the first step towards its implementation—the real difficulty is to *ensure* that the terms of agreements are complied with. We are briefly reviewing here those mechanisms that address both the specification and the enforcement of contracts.

Roscheisen and Winograd [14; 13] developed FIRM, an infrastructure used for specifying and enforcing contracts—called compacts, in their framework. The main difference between FIRM and *CARs* boils down to the disparate views we have on how contracts are to be enforced. In FIRM each contract is supported by a trusted agent which mediates between parties. Thus, the number of trusted agents needed grows *linearly* with the number of contracts and might hurt overall performance<sup>7</sup>.

Abiteboul, Vianu, Fordham and Yesha [1] introduced *transducers* (corresponding to contracts) which extend traditional relational database with rules, written in Datalog, stating the effects of an agent action (in this model, the observable events, which trigger policy evaluation, are database modifications). However, the mechanism deals with *only one* contract, and it is not clear whether several contracts can be supported in a scalable manner.

Finally, Ungureanu and Minsky [18] proposed a framework, called LGI, to support B2B e-commerce agreements. Enforcement is carried out under LGI, by associating with each communicating party, a trusted agent, called controller. The locality of enforcement solves the scalability problem: neither the number of participants in B2B transactions, nor the number of agreements is an issue any longer. However, scalability comes with a price: *all* enterprises involved have to use the same mechanism; it is still an open question whether LGI will be adopted and employed on such a large scale.

## 7. CONCLUSION

The well known principle of separation between policy and mechanism is carried out in traditional access control mechanisms by having servers enforce any policy expressible in a certain model. However, in most implementations a server is associated with a *single* policy—which has been loaded a-priori. In *CAR* framework a server is no longer bound by a unique policy: an agreement is loaded in an ad-hoc manner as the need arises. This enhanced flexibility has several important benefits, especially in the B2B e-commerce context. First, one does not have to maintain a dedicated server for each agreement (or set of agreements). As such, the number of agreements the enterprise is involved in is no longer an issue. Second, it is easy to establish new agreements: all that is required, is to deploy them on authorized repositories. Finally, in case of traffic increase, new observers can be added on the fly and without perturbing the rest of the system. To bootstrap trust, an observer needs only means for identifying valid repositories.

## 8. ACKNOWLEDGMENTS

The author is grateful to the anonymous reviewers for their helpful comments on the previous version of this paper.

## 9. REFERENCES

- [1] S. Abiteboul, V. Vianu, B. Forham, and Y. Yesha. Relational transducers for electronic commerce. In *Symposium on Principles of Database Systems*, pages 179–187, June 1998.

<sup>7</sup>Roscheisen notes in [13] that in FIRM “the number of objects ends up being quite large in realistic applications, and this quickly becomes infeasible, especially given the resources that each such object requires.”

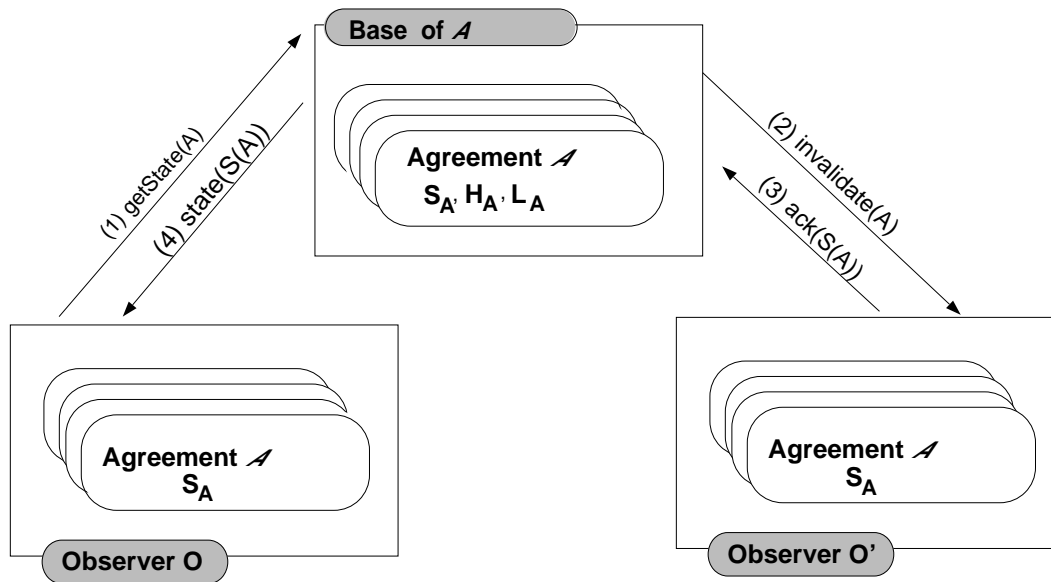


Figure 4: Distribution of the control state of a *CAR*: (1) observer  $O$  asks for  $S_A$ ; (2) the base asks  $O'$ , the current holder of  $S_A$ , to invalidate its copy (3) as a result,  $O'$  sends  $S_A$  to the base and invalidates its own copy; (4) the current version of the control state is sent to  $O$ .

- [2] X. Blanc, M. Geravis, and R. Le-Delliou. Using the UML language to express the ODP enterprise concepts. In *Proceedings of the Third International Enterprise Distributed Object Computing (EDOC99) Conference*, pages 50–59. IEEE, September 1999.
- [3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, 1603, 1999.
- [4] W. W. W. Consortium. Jigsaw - the W3C's web server. website:<http://www.w3.org/Jigsaw/>.
- [5] Economist. E-commerce (a survey). pages 6–54. (The February 26th 2000 issue).
- [6] Economist. Riding the storm. pages 63–64. (November 6th 1999 issue).
- [7] B. N. Grosf, Y. Labrou, and H. Y. Chan. A declarative approach to business rules in contracts: Courteous logic programs in XML. In *Proceedings of the first ACM Conference on Electronic Commerce (EC99)*, November 1999.
- [8] An http extension framework. Technical report. available from <http://www.w3.org/Protocols/HTTP/ietf-http-ext/>
- [9] J. Jajodia, P. Samarati, V. S. Subramanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *Proceedings ACM SIGMOD Conference on Management of Data*, May 1997.
- [10] J. Josh, W. Aref, A. Ghafoor, and E. Spafford. Security models for Web-based applications. *Communications of the ACM*, 44(2):38–44, February 2001.
- [11] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28:690–691, Sept. 1979.
- [12] B. Lampson. Protection. In *Proceedings of 5th Princeton Symposium on Information Sciences and Systems*, pages 437–443, March 1971. Reprinted in *ACM Operating Systems Revue*, Vol 8(1), pp 18-24 (Jan. 1974).
- [13] M. Roscheisen. *A Network-Centric Design for Relationship Based Rights Management*. PhD thesis, Stanford University, 1997.
- [14] M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 1996.
- [15] XML Schema. Technical report, World Wide Web Consortium. website: <http://www.w3.org/XML/Schema>.
- [16] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.
- [17] J. Ullman. *Database and Knowledge-Base Systems*. Computer Science Press, 1988.
- [18] V. Ungureanu and N. Minsky. Establishing business rules for inter-enterprise electronic commerce. In *Proc. of the 14th International Symposium on Distributed Computing (DISC 2000)*; Toledo, Spain; LNCS 1914, pages 179–193, October 2000.
- [19] Extensible markup language (XML 1.0). Technical report, World Wide Web Consortium. website: <http://www.w3.org/TR/REC-xml/>.