

An Edition Control Policy Model for Scientific Collaborative Databases

Robert Bossy

Laboratoire Informatique et Systématique
Université Pierre et Marie Curie
bossy@ccr.jussieu.fr

Abstract

The Web stimulated collaborative management of information systems. However scientific databases require seldom implemented features like validation control and data traceability. We put forward a model for designing traceable-data collaborative databases, suitable for scientific applications. The model stresses the definition of a policy of edition control for specific applications. The edition control policy (ECP) is a formal description of a collaboration model detailing the evolution of data through potential influence of contributors. ECPs have the advantage of supporting finer authorization strategies than the usual Lamson matrix. Furthermore the database documentation can be automated by keeping track of the successive contributions. We show that widely used collaborative systems, like discussion forums and peer reviewed databases, are particular instances of ECP systems. The method provides a basis for implementation of a declarative policy language processor, namely Quilombo. This software prototype takes advantage of simple RDBMS for storage and provides facilities for accessing databases on the Web. Two case studies using Quilombo are described to show feasibility and diversity of collaboration models: a free forum and a peer reviewed thesaurus.

Keywords: collaborative databases, collaboration model, edition control policy, ECP, data traceability.

1. Problems in scientific collaborative databases

The most exhaustive and successful databases available on the Web are collaborative. Several examples of scientific collaborative databases have experienced affluence, particularly in biology like the Tree of Life [11] and the Protein Data Bank [3]. Broader audience databases may also be found on the web such as the Open Directory Project[1], a collaborative web link directory. In traditional databases users can only perform read-only opera-

tions (search or browse). In collaborative databases, they can also update or add information. The collaboration implies that users are considered not only as clients but also as potential contributors[12].

Collaborative databases have the potential to solve important problems in scientific information system management. Scientific databases owe their complexity to high ratio of relationships to elementary data and -above all- to the ever changing nature of the information. As database providers are supposed to closely follow the evolution of research, a small group of maintainers is seldom able to fulfill such a task to success[13].

However one must consider an additional constraint: validation. Without validity control an open system can quickly fall into a total chaos with regard to scientific pertinence of data[15]. Pertinence is not objective but subjective, it may depend on the user's interpretation. To evaluate the pertinence of the data, additional information about origin and history of the data is required. Validation issues are not independent from data evolution since pertinence is evaluated after taking into account the whole history of different influences over the inspected data. We will call *traceability* the ability of a database to explain the existence and the state of an element by an history of events. In the case of collaborative databases, traceability means -at least- information about who contributed to each element and when. Traceable data allows the user to know more about the scientific research context (school of thought, outdated information, etc.). For that reason this information is often referred as meta-data.

Then we wish databases with apparently conflicting characteristics: collaborativeness, validity control and traceability. Collaboration urges to make the database as open as possible, but validity control requires some level of restriction. Despite the evolution ability of the collaborative database, it should not dispose of previous states of the database so that traceability requirements are satisfied. In order to handle our demands, three widely solutions are preferred on the Web:

1.1. User level restriction

Database maintainers can restrict edition access to registered users. This method has the advantage of being simple to implement and content controlled. We should point that restriction methods usually adds a level of traceability by providing the identity of the contributor for each database element.

But restricted access databases ceases to be collaborative because common users are restrained when they want to contribute because it requires to gather some information about them. For instance, if a user shows interest to contribute she/he must stop browsing the database before beginning to contribute, submit her/his registration and wait for her/his registration to be taken in account. The mechanism of the contributors community expansion is centralized, not collaborative.

1.2. Reviewing

Another method is to buffer contributions and move them permanently to the database after reviewing. This method is appealing because the update is collaborative, the contents is controlled and data can be attached to the contributor's identity. Database reviewing runs like its scientific journal counterpart, database maintainers must solicit reviewers to judge buffered contributions.

But competent experts are not always available due to lack of resources. Furthermore the reviewing task is hardly collaborative since it is the duty of database maintainers only. This reviewing mechanism shifts the bottleneck from contributors to the reviewers.

1.3. Edition control

The third solution is to restrict the possible contributions by an acute control of the database edition. Some edition control mechanisms are described as authorization models [16] or collaboration protocols [6]. We will use the term of edition control policy (ECP) so there's no possible confusion with computer protocols or limitation to authorization issues. An ECP asserts which contributions are allowed and expected given the current context of the database. The ECP described in [6] mimics a peer reviewing collaboration model, users are *allowed* to submit changes in a database and *expected* to accept or reject other users' contributions. Several other ECPs were proposed: [16], [17] and [5] based on user groups known as roles; [10] which emphasizes on synchronous collaboration (real-time conferencing). The ECPs usually work at the same rate regardless number of contributors or reviewers (if reviewing is enabled) because they do not rely on a centralized procedure at any level. The ECP method seems to answer the best to our conflicting demands but the ECPs cited above forces us to follow

more or less flexible collaboration models. Also all of them neglect traceability now that we established that our users need traceable data.

2. Traceable ECP principles

Our model emphasizes on traceability since it is the least provided feature among collaborative database tools. We also decided to extend the model into a more generic level allowing to define a wide range of different ECPs rather than just advancing yet another collaboration model. The model is based on three principles:

- contributor identities must be an integrated part of the database,
- contributions are stored rather than data, and
- the policy itself is expressed by the means of rules.

2.1. Contributors as part of data

In our model the identity of contributors is stored in the same way as the data itself inside the database. It means that there's a table that can be named `contributor`, `agent`, `author` or whatever.

Contributor identities are usually assimilated to meta-data. However meta-data can be seen as data, especially in the case of collaborative databases since we want contributions to be bound to their respective originators. So the information about the contributor may be considered for evaluation of the contribution relevance. Besides integration of contributors in the database avoids a risky and resource expensive two compound databases architecture. Also since the collaborative database is expected to grow boundlessly, contributors -as part of data- are expected to multiply in the same way.

2.2. Storing actions

The traceability becomes a problem when performing information update. Fresh data is simply added to the database but updating a record is performed by side-effecting. A new value replaces the old, which is "forgotten". This procedure forbids a full traceability since the successive contributions are lost at each update. Then we get just a snapshot of the database on a given moment while we require a full portrait of the evolution that led to the current situation. The problem is obviously the same for deletion. To solve this problem, we based our model on preserving *actions* rather than raw data. Each action represent a contribution, they are stored as *records* in classical tables. Thus an ECP may define several *action types*, each one represented by a different table. Action types must have distinctive date

and contributor fields so that each record contains respectively the action occurrence timestamp and the person accountable for the contribution. Additional fields correspond to the data on which the contribution applies.

A linguistic metaphor is useful to better understand the purpose of actions. Action types may be labeled with an active verb like `add`, `modify`, `vote` or `moderate`. So each action instance matches a simple sentence build like this:

- the verb from the action type label,
- the subject from the contributor field,
- a time complement group from the date field,
- a complement from each additional field

For instance, consider the sentence:

January the 6th 2001, John Doe added the definition “...” to the term “apomorphism”.

It may have been built from the following record:

add_definition_to_term			
date	contributor	definition	term
2001-01-06	"John Doe"	"..."	"apomorphism"
⋮	⋮	⋮	⋮

An important point is that actions are the only entry points to the database. Any contribution must fit into one of the defined action types. As a result each contributed input is linked to a date of occurrence and the identity of the contributor. And because of our traceability requirements, actions cannot be modified nor deleted. Thus a contribution is always the addition of a new action record, so the database grows continuously by cumulating contributions from various agents.

Our action based model compares to logging systems provided by many RDBMSs. However log books usually consists on flat files (log files) because this facility is assumed to be used infrequently. Contrariwise querying actions is considered routine because they are a proof of the evolution of a scientific collaborative artifact.

Our model provides a way to follow in detail the development of the database, however we still want to provide the snapshot view of the database. We can bring the whole set of action instances together in order to build a stable state corresponding to the snapshot database. This synthesizing function is performed by policy rules described below.

Obviously the model needs an authorization mechanism operating when a new action is about to be added. Authorization enforcement is also performed via policy rules. Will show now why the same element of our model is used for two distinct purposes: authorization and data calculus.

2.3. Policy rules

Rules are the third element of the model, they have two functions:

- express which actions are allowed given the current database context,
- compute the snapshot database state from stored actions.

There must be a rule for each action type, it is called the action’s *authorization rule*. An elementary contribution consists of a new action instance, thus containing the current date and the current user. Each new action instance must satisfy positively the rule of the corresponding action type, then authorization rules allow or reject new action instances.

The most used authorization model is represented by a table of permissions known as a Lampson matrix[9]. Each row is labelled with a contributor identification and each column by an operation, the matrix is filled by booleans so it states who has the right or not to perform each operation. Action’s authorization rules can emulate a Lampson matrix by mentioning the contributor. However this model may be inadequate because it is not sensitive to the contents of the database. Our model is able to express more complex authorization strategies by writing rules that examine some pieces of the database. For instance, let a system that allows users to select between alternative solutions to a problem by voting. We will define (amongst other) two actions: `submit_problem`, `submit_alternative` and `vote`. It is possible to impose deadlines by stating the following authorization rules:

- `submit_alternative` to a problem is allowed **if and only if** the current date is less than a week after the corresponding `submit_problem` instance was committed.
- `vote` is allowed **if and only if** the current date is between one and two weeks after the corresponding `submit_problem` instance was committed **and** the contributor did not vote yet for an alternative to this problem.

This rule set results in the following action sequence:

1. A contributor submits a problem (commits a `submit_problem` instance).
2. For the next week, contributors are expected to submit alternatives to a problem (commit `submit_alternative` instances).
3. For the following next week, contributors vote for the alternative of their choice.

Another set of rules is used to produce a snapshot view of the database. They are called *volatile rules* because they may yield different results from an instant to another, depending on contributions occurrences in the meanwhile. Since deletion and modification is banned from our model, volatile rules may take in account: modifying actions to show the correct state of an element of the database, deletion actions to hide data elements. Moreover they can also consider moderation actions to give information about pertinence. If we carry on with the voting system, we can define `score` that links a `submit_alternative` instance to a number and `prevail` that links a `submit_problem` instance to a `submit_alternative` instance:

- `score` links a `submit_alternative` instance -noted A - to a number -noted N_A - **if and only if** N_A is the cardinality of the set of `vote` instances referring to A .
- `prevail` links a `submit_problem` instance -noted P - to a `submit_alternative` instance A **if and only if** the `score` of A is the higher amongst all `submit_alternative` instances referring to P .

3. ECPs case studies

We exposed a model made of three parts (contributors, actions and rules) in terms of general guidelines. Elements of each part are connected, so assembling them to shape a meaningful policy calls for a little practice. A step by step recommendation might be helpful in the process of building a special purpose ECP:

1. **Goals:** What is the intended audience? What data is to be provided?
2. **Collaboration model:** Which kinds of interaction between users? How many people will be involved? Which data will be reviewed, moderated, rectifiable?
3. **Contributors:** What is the intended audience (again)? Who/where are desirable contributors? What information about contributors is relevant?
4. **Actions:** In what consist contributions? What contributors will be able to affect in the database? In which conditions or prerequisites?
5. **Volatiles:** Which information is to be provided (again)? Which actions may exhibit new relevant data?

In order to illustrate this recipe, we will now describe two different ECPs in detail for a better understanding of our model.

3.1. Free forum

The free forum ECP describes the collaboration model of Internet based forums: users send messages and reply to previous messages. We want a wide open system so everybody can post a message. Contributors may provide their identity (first name and last name) or post messages anonymously. Thus we define a `contributors` table with the following fields:

- `contributor_id`, the primary key.
- `first_name`, a string containing the first name.
- `last_name`, a string containing the last name.

Now we define the only possible action type named `post_message`, the corresponding table is declared with the fields:

- `post_message_id`: the primary key.
- `date`: the date the action instance was committed.
- `contributor`: a foreign key referencing the identity of the contributor, a NULL value means the message was posted anonymously.
- `title`: a string containing the message title.
- `body`: a large string containing the message body.
- `reply`: a foreign key referencing the message this one replies to, a NULL value means the message does not reply to another one.

The `date` and `contributor` fields are mandatory (*cf.* 2.2). Each `post_message` record can be interpreted, by replacing field names by the actual values, as a sentence:

At `date`, `contributor` posted a message entitled `title` in reply to `reply` saying “`body`”.

The authorization rule for `post_message` is trivial since there is actually no rule, everybody can post any message at any time. Thus contributors can post messages by adding new `post_message` records.

Then we look for relevant information that can be computed from `post_message` instances. We first define the `thread volatile` rule:

A message is a `thread` **if and only if** it replies to no other message (a `post_message` record which `reply` field has a NULL value).

We also want to know which messages are involved in a thread, we achieve this with the `involves volatile` rule defined as:

A thread T involves a `post_message` instance P **if and only if** P replies to T **or** P replies to X **and** T involves X .

This rule recursively yields all messages replying directly or not to a single thread. However fetching active threads is even more useful, we also define the `active` rule that computes threads involving recent messages:

A thread T is `active` **if and only if** there is a `post_message` P such as T involves P **and** P is less than a week old.

Now we get a formal definition of a simple forum collaboration model describing the interactions between users and their consequences. However forum has quite a lot of variations found over the Internet. Anonymous contribution may be forbidden or just restricted, for instance we can state that anonymous users cannot start a new thread. Messages could also be moderated by adding a new `moderate` action type and make a few changes:

- a new `messages_ok` volatile rule that yields moderated messages,
- the authorization rule for `send_message` states that one cannot reply to a non moderated message, and
- a new set of actions and volatile rules to designate moderators.

Moderation clearly complicates the policy definition, but the free forum has been made deliberately elementary. The next example enables a similar mechanism to moderation: peer reviewing.

3.2. Peer reviewed thesaurus

The peer reviewed thesaurus defines an ECP for domain specific thesaurus collaborative databases. A thesaurus database will contain actions on terms and definition of terms. The intended audience is any expert of the specific domain. Users will be able to submit terms and definitions or correct a definition but submissions should be visible only after reviewing. The `contributors` table is exactly the same as the forum one (cf. 3.1), however users won't be able to contribute anonymously.

We define the submission action types: `submit_term`, `submit_definition`, `correct_definition`. These actions are used by contributors to add new terms, definitions and definition corrections respectively. Since `date`, `contributor` and primary key fields are implied, we only detail the authorization rule and additional fields:

- `submit_term`, each record is a term submission:
 - `term`, a string naming the term.

iff term is not already an accepted term.

- `submit_definition`, each record is a submission of a term definition:
 - `term`, a foreign key referencing the corresponding `submit_term` record, this indicates to which term this definition applies.
 - `definition`, a large string containing the definition.

iff term is an accepted term.

- `correct_definition`, each record is a rectification of a term definition:
 - `initial_definition`, a foreign key referencing the corresponding `submit_definition` record, this indicates which definition is to be corrected.
 - `new_definition`, a large string containing the correct definition.
- iff** `initial_definition` is an accepted definition **and** there is no other unreviewed correction submission for this definition.

We also define the analogous reviewing actions: `review_term`, `review_definition` and `review_correction`. Besides primary key, `date` and `contributor` fields, they contain a field indicating the submitted item reviewed. This field is named `term_submission`, `definition_submission` and `correction_submission` respectively referencing a `submit_term`, a `submit_definition` or a `correct_definition`. In addition all reviewing actions have a `score` integer field containing of reviewer's evaluation: we take as convention that a negative value is a reject and a positive value is an accept. The authorization rule is the same for the three reviewing action:

A contributor C can review X **iff** C is a peer **and** X has not been reviewed yet.

We note a term has possibly several definitions, and each definition can evolve through corrections. Another interesting fact is that a rejected term may be re-submitted later by the same contributor (or another) until it is accepted. In this ECP, volatile rules have a more important role than the forum because they tell us the result of contributors' actions. In authorization rules, we mentioned "accepted terms", now we define the `term` rule that produces accepted terms:

T is a term **iff**
there is a `submit_term` record S_T such as
 T equals the `term` field of S_T **and**
there is a `review_term` record R_{S_T} such as S_T

equals the term field of R_{S_T} **and**
the score field of R_{S_T} is positive.

In other words, T becomes term after the sequence of actions:

1. a contributor submits S_T
2. a peer reviews and accepts S_T

→ T is a term

We define a similar volatile rule definition that yields the accepted definitions. However for each definition we must search for the most recent correction, so we define correct:

D_n is the correct string of definition D
iff
if there is no accepted correct_definition record which initial_definition field equals D
then D_n equals the definition field of D
else D_n equals the new_definition field of the last accepted correct_definition record.

The correct rule tells us the final text of a definition taking in account the last correction. The following sequence shows the successive texts for a single definition:

1. a contributor submits the definition D_0
2. a peer accepts D_0
3. a contributor submits a correction D_1 to D_0
4. a peer accepts D_1
5. a contributor submits a correction D_2 to D_0
6. a peer accepts D_2
7. et cetera until D_n

At stage 2, the correct definition text is the definition field contents of D_0 , at stage 4 the new_definition field contents of D_1 and finally, at current date, the correct definition is the new_definition field contents of D_n . Now we can produce terms, several definitions for each terms and the correct text for each definition. Furthermore each element is guaranteed to be peer reviewed. Hitherto we distinguished normal contributors from peers, at least we define peer as a volatile rule:

A contributor P is a peer **iff** he submitted an item (submit_term, submit_definition, correct_definition or submit_crossref) accepted by a peer.

The clever reader will notice that nobody is able to review the first submitted item since there are no peers. We leave the clever reader patch this problem as an exercise, one solution relies on adding a fixpoint to the peer rule. The thesaurus ECP also have several variants, for instance one can add actions to work on cross reference between definitions (synonyms, related ...). A righteous change to reviewing action authorization rules should state that a peer cannot accept his own submissions.

Both sample ECPs shows how generic our model can be, however the rules mentioned above must be expressed more formally to allow actual implementation. So the model provided us a basis for a ECP specification language allowing a formal declaration of ECP rules.

4. Quilombo software overview

Someone willing to provide a collaborative information system matching our requirements may use our model as guidelines for a database conception. Action types and data can be straightforward implemented as simple tables, however there are a wide range of alternatives to implement rules. Depending of the RDBMS the provider choose, rules can be implemented through temporary tables, triggers, views, stored procedures and builtin authorization mechanisms. But the provider fails to gather the ECP into a single place as an explicit rule set possibly leading users to misunderstand the control policy. Additionally *ad hoc* ECP implementations lack portability, because technical choices may be improper in another RDBMS context.

We implemented a software called *Quilombo* with the purpose of writing ECPs and animate them in connection to a RDBMS. Our goal here is not to give the software full specification and usage, for sources and documentation are freely available¹. *Quilombo* is made of two main pieces `qgen` and `qquery` used respectively for creating and querying a collaborative database. *Quilombo* uses a standard relational database as back end storage so the data still available with the usual SQL tools. The `qgen` utility reads and checks an ECP specification and eventually generates SQL code for the storage database creation (thus the SQL code consists on a series of CREATE TABLE expressions). The `qquery` utility solves *Quilombo* queries by fetching persistent data from the relational database and by interpreting the ECP rules.

Authorization and volatile rules are expressed in a logic-based language, thus the *Quilombo* language is a *Prolog* variant. Queries are also logic-based so the engine can extract data directly from the relational database and compute data from volatile rules. Predicate logic intrinsically expresses rules and translates quite straightforwardly to

¹<http://lis.snv.jussieu.fr/quilombo>

SQL. However it has powerful features: for instance we mentioned a recursive rule in the forum ECP (*cf.* 3.1) but SQL actually cannot express recursive queries, contrastably the logic programming paradigm encourages recursion. *Quilombo* queries provides reading access to the snapshot view as well as the whole set of actions. The system does not evaluate the whole snapshot view, it only computes required parts from volatile rules at query-time.

Additionally a special syntax is provided to add new action instances. Naturally the engine enforces authorization rules by checking that each instance satisfies the corresponding authorization rule before operating the actual storing operation.

The ECP specification language recognize four distinct declarations: `data`, `contributor`, `action` and `volatile`. The `data` declaration indicates tables of raw data, the one representing contributors is indicated by a `contributors` declaration. Each `action` declaration describe an action type, its additional fields label and types as well as the authorization rule. Finally the `volatile` declaration expresses volatile rules.

By all means the user must possess knowledge about the ECP in order to contribute efficiently. Web interfaces are established and practical tools for accessing databases, especially collaborative databases. The *Quilombo* package also contains a special utility -namely `qtag`- to query ECP systems through simple CGI.

5. Conclusions

Our model allows conception of databases edited in a collaborative mode by the means of an edition control policy. The ECP has three components:

- The description of **contributors**,
- Contributions to the database are categorized through **actions types**,
- **Rules** form an authorization strategy and data compiler.

The usage of ECP supposes that traceability is crucial, the volatile rules being an explicit explanation of the expected history of the data. The model is aimed to scientific applications especially in history sensitive disciplines such as biology.

For instance, there is a project to define an ECP for the biological nomenclature —the science of naming species—. Indeed the same name may indicate different species from a time to another due to the evolution of the biological knowledge (eg. taxonomies). However the community biologists regularly edit nomenclature codes[8][7] in order to

provide stability in the names. These codes assert sophisticated rules of naming all biological publication must comply with. So with an extensive awareness of anterior naming acts found in the literature, one is able to discover the right meaning of a name at a given date. The ECP method suits better to nomenclature than existing information systems management[2] and an ECP based database will hopefully help biologists to keep names in strict observance to the codes.

The specification of an ECP requires a contribution oriented design, thus making databases ready for collaborative edition since the conception of the schema. Each ECP suits to a particular application so the model is able to express a variety of different collaboration models so the model meets the generality requirements.

The *Quilombo* prototype proves that the model is implementable and that collaborative databases are workable. Future implementations can take advantages of database research advances. The logic based queries makes *Quilombo* comparable to deductive databases. Indeed actions and data are extensional relations while volatile and authorization rules are intensional relations[14]. Also, since all records have a timestamp, it is also comparable to temporal databases[4]. Deductive and temporal database are active research domains producing efficient query processing algorithms.

References

- [1] The open directory project. Internet address: <http://www.dmoz.org>.
- [2] N. Bailly and J.-C. Hureau. Bases de données en biologie: quelques problèmes liés à la nomenclature et aux références bibliographiques. *Cybiurn*, 19(4):333–342, 1995.
- [3] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, (28):235–242, 2000.
- [4] J. Chomicki and D. Toman. Temporal logic in information systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 3, pages 31–70. Kluwer, 1998.
- [5] W. K. Edwards. Policies and roles in collaborative applications. In *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work*, Language Support for Groupware, pages 11–20, 1996.
- [6] J. Euzenat. A protocol for building consensual and consistent repositories. Technical Report RR-3260, Institut National pour la Recherche en Informatique et en Automatique, Grenoble (France), September 1997.
- [7] W. Greuter, J. McNeill, F. R. Barrie, H. M. Burdet, V. Demoulin, T. S. Filgueiras, D. H. Nicolson, P. C. Silva, J. E. Skog, P. Trehane, N. J. Turland, and D. L. Hawksworth, editors. *International Code of Botanical Nomenclature (Saint Louis Code)*. Koeltz Scientific Books, Königstein, 2000.

- [8] International Commission for Zoological Nomenclature. *International Code of Zoological Nomenclature*. International Trust for Zoological Nomenclature, fourth edition, 1999.
- [9] B. W. Lampson. Protection. *Operating Systems Review*, 8(1):18–24, Jan. 1974.
- [10] D. Li and R. R. Muntz. A collaboration specification language. *ACM SIGPLAN Notices*, 35(1):149–162, 2000.
- [11] D. R. Maddison and W. P. Maddison. The tree of life: A multi-authored, distributed internet project containing information about phylogeny and biodiversity. Internet address: <http://phylogeny.arizona.edu/tree/phylogeny.html>, 1998.
- [12] P. Marty. Museum informatics and collaborative ontologies: the emerging socio-technological dimension of information science in museum environments. *Journal of the American Association for Information Science*, 50(12):1083–1091, 1999.
- [13] G. D. Michelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopoulos, M. Papazoglou, K. Pohl, J. Schmidt, C. Woo, and E. Yu. Cooperative information systems: A manifesto. In M. P. Papazoglou and G. Schlageter, editors, *Cooperative Information System: Trends and Directions*. Academic Press, 1997.
- [14] R. Ramakrishnan and J. D. Ullman. A survey of research on deductive database systems. *Journal of Logic Programming*, 23(2):125–149, 1993.
- [15] H. Shen and P. Dewan. Access control for collaborative environments. In *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work*, Building Real-Time Groupware, pages 51–58, 1992.
- [16] K. Sikkel. A group-based authorization model for cooperative systems. In Kluwer, editor, *Proc. European Conference on Computer-Supported Cooperative Work (ECSCW'97)*, pages 345–360, September 1997.
- [17] K. Sikkel and O. Stiemerling. User-oriented authorization in collaborative environments. In *Proc. 3rd International Conference on the Design of Cooperative Systems (COOP'98)*, volume 1, pages 175–183, Cannes, May 1998.