W Prinz, M Jarke, Y Rogers, K Schmidt, and V. Wulf (eds), *Proceedings of the Seventh European Conference on Computer-Supported Cooperative Work, 16-20 September 2001, Bonn, Germany*, pp 59-77
© 2001 Kluwer Academic Publishers Printed in the Netherlands

Team Automata for Spatial Access Control

Maurice H. ter Beek
LIACS, Universiteit Leiden, The Netherlands

Clarence A. Ellis
Department of Computer Science, University of Colorado, U.S.A.

Jetty Kleijn
LIACS, Universiteit Leiden, The Netherlands

Grzegorz Rozenberg
LIACS, Universiteit Leiden, The Netherlands
and
Department of Computer Science, University of Colorado, U.S.A.

Abstract. Team automata provide a framework for capturing notions like coordination, collaboration, and cooperation in distributed systems. They consist of an abstract specification of components of a system and allow one to describe different interconnection mechanisms based upon the concept of "shared actions" This document considers access control mechanisms in the context of the team automata model It demonstrates the model usage and utility for capturing information security and protection structures, and critical coordinations between these structures On the basis of a spatial access metaphor, various known access control strategies are given a rigorous formal description in terms of synchronizations in team automata

Introduction

As the complexity of technical systems continues to increase, abstractions tend to be especially useful. For this reason, computer science often introduces and studies various models of computation that allow enhanced understanding and analysis Computer science has also created a number of interesting metaphors (e.g , the desktop metaphor) that aid in end user understanding of computing phenomena. This document is concerned with a model and a metaphor. The model is team automata, which were created explicitly for the specification and analysis of CSCW phenomena and collaborative systems (Ellis, 1997). The metaphor is spatial access control, which is based upon current notions of virtual reality, and helps demystify concepts of access control matrices and capability structures for the end user (Bullock et al., 1999).

Many of the concepts and techniques of computer science, such as concurrency control, user interfaces, and distributed databases, need to be rethought in the groupware domain. Team automata are helpful for this rethinking. The framework provided by the team automata model allows one to separately specify the components of a collaborative system and to describe their interactions. It is neither a message passing model nor a shared memory model, but a shared action model. It has been proposed as a formal framework for modeling both the conceptual and the architectural level of groupware systems (Ellis, 1997) Components can be combined in a loose or more tight fashion depending on which actions are to be shared, and when. Such aggregates of components can then in turn be used as components in a higher-level team. Thus team automata fit nicely with the needs and the philosophy of groupware (Ellis, 1997) and thanks to the formal setup, theorems and methodologies from automata theory can be applied. Team automata are an extension of Input/Output automata (Lynch, 1996) and are related to, but different from Vector Controlled Concurrent Systems (Keesmaat, 1996), Petri nets (Reisig et al , 1998), and other models of concurrent and collaborative systems (Nutt, 1997).

Our spatial access control metaphor piggy-backs upon the virtual reality metaphors of places and spaces (Bullock, 1998) Different places are conducive to different activities, and different rooms and different buildings have different affordances. The metaphor of virtual rooms and virtual buildings can help to guide the user through a complex computer system to find the resources needed for a particular task. In need of a certain document, e.g., the user would naturally think of entering a virtual library, where he or she would have read access.

In the following sections we first discuss the spatial access control metaphor by means of an example and subsequently gently present the team automata model by applying it to this example In the core of the document we then show how certain spatial access control mechanisms can be made precise and given a formal description using team automata. First we introduce information access modeling by granting and revoking access rights, and show how immediate versus delayed revocation can be formulated. In the subsequent section we extend our study to the more complex issue of meta access control, and consequently we show how team automata can deal with deep versus shallow revocation.

The style of this document is relatively informal. Full formal definitions, observations, and results relating to team automata can be found in (ter Beek et al., 1999). Our aim here is to connect the metaphor of spatial access control to the framework of team automata, and to show through examples how this combination facilitates the identification and unambiguous description of some key issues of access control. The rigorous setup of the framework of team automata allows one to formulate, verify, and analyze general and specific logical properties of various control mechanisms in a mathematically precise way. In realistically large systems, security is a big issue, and team automata allow formal proofs of correctness of its design. Moreover, a formal approach as provided by the team automata framework forces one to unambiguously describe control policies and it may suggest new approaches not seen otherwise. There is a large body of literature concerning topics like security, protection, and awareness in CSCW systems. Although team automata are potentially applicable also to these areas, this paper is not concerned with issues outside of spatial access control. In the final section we discuss some variations and extensions of our setup.

Access Control

A vital component of any system or environment is security and information access control, but this is sometimes done in a rather ad hoc or inadequate fashion with no underlying rigorous, formal model. In typical electronic file systems, access rights such as read-access and write-access are allocated to users on some basis such as "need to know", ownership, or ad hoc lists of accessors. Within groupware systems, there are typically needs for more refined access rights, such as the right to scroll a document that is being synchronously edited by a group in real time. Furthermore, the granularity of access must sometimes be more fine-grained and flexible, as within a software development team. Moreover, it is important to control access meta-rights. For example, it may be useful for an author to grant another team member the right to grant document access to other non-team members (i.e. delegation). Various models have been proposed to meet such requirements (see, e.g , (Shen et al., 1992), (Rodden, 1996), and (Sikkel, 1997)).

We use a spatial access metaphor based upon recent work of Bullock and colleagues in (Bullock et al., 1997) and (Bullock et al., 1999). There, access control is governed by the rooms, or spaces, in which subjects and objects reside, and the ability of a subject to traverse space in order to get close to an object. Bullock also implemented a system called SPACE to test out some of these ideas (Bullock, 1998). A basic tenet of the SPACE access model is that a fundamental component of any collaborative environment is the environment itself (i.e. the space). It is the shared territory within which information is accessed and interaction takes place. Often this shared space is divided into numerous regions that segment the space. This allows decomposition of a very large space into smaller ones for manageability. It also allows cognitive differentiation (i.e. different concerns, memories, and thoughts associated with different regions), and distributed implementation (i e. different servers for different regions).

By adopting a spatial approach to access control, the SPACE metaphor exploits a natural part of the environment, making it possible to hide explicit technical security mechanisms from end users through the natural spatial makeup of the environment. These users can then make use of their knowledge of the environment to understand the implicit security policies Users can thus avoid understanding technical concepts such as so-called access matrices, which helps to avoid misunderstandings.

We consider here a virtual reality, in which a user can traverse from room to room by using keyboard keys, the mouse, or fancier devices It is a natural and simple extension to assume that access control checking happens at the boundaries (doors) between spaces (rooms) when a user attempts to move from one room to another. If the access is OK, then the user can enter and use the resources associated with the newly entered room.

To illustrate the various concepts throughout this document, we present a simple running example which is concerned with read and write access to a file F by a user Kwaku This file might be any data or document that is stored electronically within a typical file system. The file system keeps track of which users have which access rights to the file F. Three types of access rights are possible for a file F: null access (implying the user can neither read nor write the file), read access (implying the user cannot write the file), and full access (implying the user can read and write — i.e. edit — the file).

In security literature, authentication deals with verification that the user is truly the person represented, whereas authorization deals with validation that the user has access to the given resource. Assume that when Kwaku logs into the system, there is an authentication check. Then whenever he tries to read or write F, authorization checking occurs, and Kwaku is either allowed the access, or not. Using the SPACE metaphor, the above three types of access rights can be associated with three rooms as shown in Figure 1.

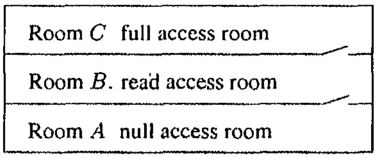
Room C full access room
Room B. read access room
Room A null access room

Figure 1 A rooms metaphor for access control

Room A is associated with no access to the document, room B is associated with read access, and room C models full access Suppose Kwaku is in room B, the reading room. Presence in this room means that any time Kwaku decides to read F, he can do so. However, if he attempts to make changes to F, then he will fail because he does not have write access in room B. There are doors between rooms, implying that user access rights can be dynamically changed by changing rooms. We discuss this dynamic change in more detail in a later section of this document.

This access mechanism satisfies a number of end user friendly properties: it is simple, understandable by non-computer people, relatively natural and unobtrusive, and elegant. In the next sections we show how modeling this type of access metaphor via team automata adds precision, mathematical rigor, and analytic capabilities.

Team Automata

In this section we introduce component automata and team automata as formally defined in (ter Beek et al., 1999) by using the example of the previous section.

A team automaton consists of component automata, combined in a coordinated way such that they can perform shared actions. Component automata within a team automaton can during each clock tick simultaneously participate in one instantaneous action (i.e synchronize on this action), or remain idle. The team automata model forms a mathematical automata theoretic specification, rather than a high-level language specification such as Hoare's CSP (Brookes et al., 1984), Forman's Raddle (Evangelist et al., 1988), or the n-party interaction mechanism (Attie et al., 1990). Team automata, like I/O automata, are adequate for specifying shared memory systems and message passing systems, although they are neither of the two While inappropriate for capturing aspects of group activity such as social aspects and informal unstructured activity, the model has proved useful in various CSCW modeling areas (Ellis, 1997). A spectrum from hardware components to interacting groups of people can be modeled by team automata.

The component automata are rather ordinary, but their interconnection strategy is intriguing because, as we mentioned, it is neither shared variable nor message passing. We classify the actions which take an automaton from one state to another into two main categories, one of which is subdivided into two more categories. *Internal* actions have strictly local visibility and can thus not be observed by other components, whereas *external* actions are observable by other components. These external actions are used for communication between components and consist of *input* actions and *output* actions Composing component automata into team automata is based on an interconnection strategy of shared actions, in which one or more automata participate in the execution of the same external action (which may be input to some components and output to other components). The choice for a specific interconnection strategy is based on what one wants to model, and this possibility to choose is the main feature of the team automata framework

We now return to our access control example by showing how to model it in the team automata framework. The component automaton M^C depicted in Figure 2(a) corresponds to room C of Figure 1, as it models full access to file F. The states of M^C are C_e modeling an empty room, C_n modeling F is not accessed, C_r modeling F is being read, and C_w modeling F is being written (edited). The wavy arc in Figure 2(a) denotes the initial state C_e. The actions of M^C are e_{BC} (enter room), e_{CB} (exit room), r^C (begin reading), $\underline{r}^C$ (end reading), w^C (begin writing), and $\underline{w}^C$ (end writing).

A component automaton thus consists of *states*, *actions*, and (labeled) *transitions* which describe state changes caused by actions We distinguish a set of *initial states* and the set of actions is further partitioned into *input*, *output*, and *internal* actions. Hence a component automaton is a labeled transition diagram with three distinguished types of labels (actions).

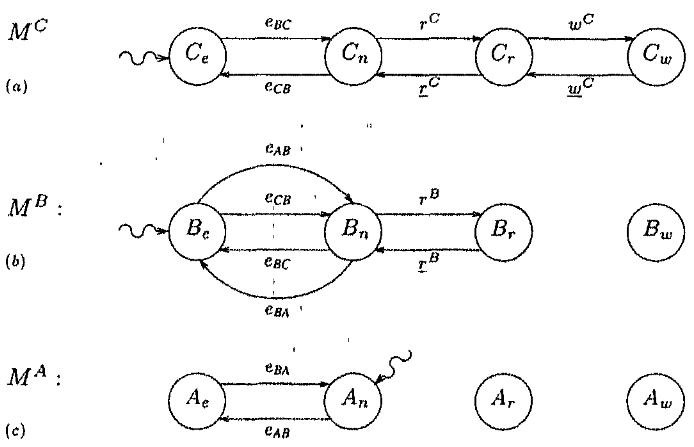

Figure 2 Automata M^C, M^B, and M^A rooms C, B, and A

Returning to M^C we have the transitions (C_e, e_{BC}, C_n), (C_n, e_{CB}, C_e), (C_n, r^C, C_r), $(C_r, \underline{r}^C, C_n)$, (C_r, w^C, C_w), and $(C_w, \underline{w}^C, C_r)$. Now the transition (C_e, e_{BC}, C_n), e.g , shows that in M^C we can go from state C_e to C_n by executing action e_{BC}. We also see that transitioning directly from C_n to C_w is not possible. Furthermore, entering and exiting room C may only occur via state C_n. We choose to specify actions r^C, $\underline{r}^C$, w^C, and $\underline{w}^C$ as internal actions of M^C, and e_{BC} and e_{CB} as external actions of M^C. Both e_{BC} and e_{CB} clearly should be externally visible and therefore cannot be internal. For the moment we choose them to be output actions These two external actions are candidates for being synchronized with actions of the same name in other component automata when we form a team automaton over M^C and the two component automata as described next.

Component automata M^B and M^A corresponding to rooms B and A, respectively, are somewhat similar to M^C. However, write access is denied in rooms B and A and read access is denied in room A. Automata M^B and M^A are depicted in Figure 2(b,c). Note that M^A has initial state A_n (hence initially room A is not empty) and that both M^B and M^A have states unreachable from the initial state Actions r^B and $\underline{r}^B$ are internal, while the rest of the actions of M^B and M^A are external (output) actions.

Now suppose that we want to combine M^C, M^B, and M^A into one (team) automaton reflecting a given access policy. Then, first of all, the internal actions of each of these components should be private, i.e. uniquely associated to one component automaton. This is formally expressed by stating that when composing a team from a collection S of component automata, no internal action of any component automaton from S may appear as an action in any of the other component automata in S If this is the case, then S is called a *composable system.*

The three automata in our example clearly form a composable system and we combine them into a team automaton T^{CBA} as follows. Each state of T^{CBA} is a combination of a state from M^C, a state from M^B, and a state from M^A (hence T^{CBA} has upto $4^3 = 64$ states). Initially T^{CBA} is in state (A_n, B_e, C_e), a combination of initial states from the three component automata This means one starts in room A, while room B and C are empty.

Assuming that one can have only one kind of access rights at a time, two of the rooms should be empty at any moment in time This means that T^{CBA} should be defined in such a way that in each of its reachable states two of the three automata are always in state "empty". We let the automata synchronize on the external actions e_{AB}, e_{BA}, e_{BC}, and e_{CB}. Each such synchronized external action of T^{CBA} corresponds to exiting a room while entering another. Synchronization of action e_{AB}, e.g., models a move from room A to room B. This move is represented by the transition $((A_n, B_e, C_e), e_{AB}, (A_e, B_n, C_e))$ showing that in automaton M^A we exit room A, in automaton M^B we enter room B, and in automaton M^C we do nothing (i.e. remain idle). This represents a change in access rights from null access (in room A) to read access (in room B). We do not include, e g., the transition $((A_n, B_e, C_e), e_{AB}, (A_e, B_e, C_e))$ which would let the user exit room A but never enter room B. Furthermore, the user could be in more than one room at a time if we would allow transitions like $((A_n, B_e, C_e), e_{AB}, (A_n, B_n, C_e))$. In T^{CBA} we include only the four transitions representing the synchronized changing of rooms. In each of these transitions, one automaton is idle All internal (read and write related) actions are maintained. In each of these only that component is involved to which such an action belongs.

The reachable part of the thus defined T^{CBA} is depicted in Figure 3.

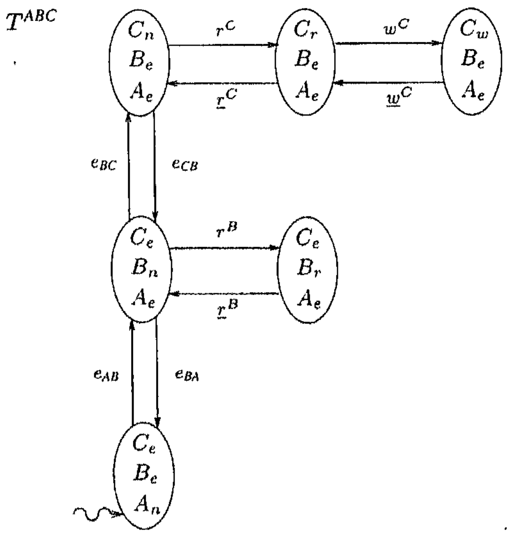

Figure 3 Team automaton T^{CBA} over M^C, M^B, and M^A

At this moment it is important to stress that T^{CBA} is not the only team automaton over M^C, M^B, and M^A. In general, there is no *unique* team automaton over a composable system $\mathcal{S}$, but a framework is provided within which one can construct a variety of team automata over $\mathcal{S}$ The flexibility lies in the choice of the transition relation for a team automaton over $\mathcal{S}$, which is based on but not fixed by the transition relations of the component automata in $\mathcal{S}$. The requirements the transition relation of a team automaton has to satisfy are as follows.

The *complete transition space* of action a in $\mathcal{S}$ consists of all transitions on a from a state q to a state q' of the team such that at least one component automaton is active, i.e. performs a. Moreover, each of the component automata either also executes a (i.e. joins in executing a) or remains idle (thus does not change state). Consequently, the transformation of the state of the team automaton is defined by the local state changes of the components involved in the action that is executed. The transitions in the complete transition space (of a) are referred to as *synchronizations* (on a) For each action a a specific subset of its complete transition space is chosen. In the case of an internal action however, each component retains all its possibilities to execute that action and change state. Note that since $\mathcal{S}$ is a composable system, synchronizations on internal actions never involve more than one component.

Any choice of a transition relation satisfying these requirements defines a team automaton over $\mathcal{S}$. Its states, initial states, as well as a partition of its actions are fixed (and the same for all team automata over $\mathcal{S}$). The state space of any team automaton $\mathcal{T}$ over $\mathcal{S}$ is the product of the state spaces of the component automata of $\mathcal{S}$, with the products of their initial states forming the initial states of $\mathcal{T}$. The internal actions of the components are the internal actions of the team automaton. Each action which is output for one or more of the component automata is an output action of the team automaton. Hence an action that is an output action of one component and also an input action of another component, is considered an output action of the team. The input actions of the component automata that do not occur at all as an output action of any of the component automata, are the input actions of the team The reason for constructing the alphabet sets of a team automaton from the alphabet sets of the component automata in the way described above, is based on the intuitive idea of (Ellis, 1997) that when relating an input action a of a component automaton to an output action a of another component, the input may be thought of as being caused by the output. On the other hand, the output action remains observable as output to other automata As shown in (ter Beek et al., 1999), every team automaton is again a component automaton and hence can be used in a higher-level team.

In $\mathcal{T}^{CBA}$, as mentioned before, the decision to consider e_{AB}, e_{BA}, e_{BC}, and e_{CB} as output actions in all component automata was made more or less arbitrarily. In fact, it depends on how one views the action of entering and exiting a room within the team automaton $\mathcal{T}^{CBA}$. By choosing all of those actions to be output (and thus of the same type), exiting one room and entering another is seen as a *collaboration* between peers.

In (ter Beek et al., 1999), where different types of synchronizations on actions shared between components of a team are classified, this synchronization of external actions of the same type is called a *peer-to-peer synchronization*. On the other hand, *master-slave synchronization* occurs when input actions *cooperate* with output actions. In that case, input can only occur as a response (slave) to output.

In our example, assume that one views the changing of rooms as an action initiated by leaving a room and forcing the room that is entered to accept the entrance. Then one would name, e g., e_{AB} an output action of M^A and an input action of M^B, and e_{BA} an output action of M^B and an input action of M^A. This causes e_{AB} to be a master-slave synchronization between master M^A and slave M^B and e_{BA} to be a master-slave synchronization between master M^B and slave M^A Likewise for the other actions.

In addition, (ter Beek et al., 1999) defines strategies that lead specifically to uniquely defined peer-to-peer and master-slave combinations within team automata. The team automata framework allows one to model many other features useful in virtual reality environments. A door, e.g., can be extended to join more than two rooms since any number of automata can participate in an output action. Furthermore, as said before, a user could be in more than one room at a time.

Authorization and Revocation

We continue our example of the previous section by adding Kwaku, a user whose access rights to file F will be checked by the access control system T^{CBA} of Figure 3. Kwaku is represented by automaton M^U, depicted in Figure 4 This extension complicates our example in the sense that Kwaku's read and write access rights can be changed independently of his whereabouts Only to enter a room he has to be authorized Thus access rights are no longer equivalent with being in a room, but rather with the possibility to enter a room. To add this to the team automaton formalization, we will use the feature of iteratively constructing teams with teams as components.

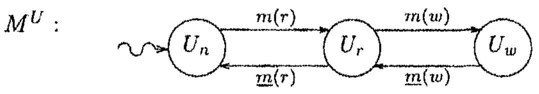

Figure 4 Automaton M^U user Kwaku

Kwaku starts in state U_n with no access rights. The actions $m(r)$, $\underline{m}(r)$, $m(w)$, and $\underline{m}(w)$ model the (meta) operations of "being granted read access", "being revoked read access", "being granted write access", and "being revoked write access", respectively Since these clearly are passive actions from Kwaku's point of view, we choose all of them to be input actions. Note that Kwaku can end up in state U_w if and only if he was granted access rights to read and to write, i.e. actions $m(r)$ and $m(w)$ have taken place. When Kwaku's write access is consequently revoked by transition $(U_w, \underline{m}(w), U_r)$, he ends up in state U_r.

Now suppose that we want to model Kwaku's options for editing file F, which is protected by the access control system T^{CBA}. Then we would like to compose a team automaton over T^{CBA} and M^U. To do so, first note that T^{CBA} and M^U form a composable system. Next we choose a transition relation, i.e. for each action a subset from its complete transition space in T^{CBA} and M^U is selected, thereby formally fixing an access control policy for Kwaku under the constraints imposed by T^{CBA}.

The initial state of any team over T^{CBA} and M^U is (A_n, B_e, C_e, U_n), i.e Kwaku is not yet editing F and is in the virtual room A without access rights Now imagine the access rights to be keys Hence Kwaku needs the right key to enter reading room B, i.e. action $m(r)$ must take place before action e_{AB} becomes enabled This action $m(r)$ leads us from the initial state to (A_n, B_e, C_e, U_r). Now Kwaku has the key to enter room B by $((A_n, B_e, C_e, U_r), e_{AB}, (A_e, B_n, C_e, U_r))$. This transition models the acceptance of Kwaku's entrance of room B, i.e. this action is the authorization activity mentioned earlier Hence our choice of the transition relation fixes the way we deal with authorization. Including, e.g., $((A_n, B_e, C_e, U_n), e_{AB}, (A_e, B_n, C_e, U_n))$ in the transition relation would mean that Kwaku can enter room B without having read access rights for F. Note however that since transitions involving internal actions of either T^{CBA} or M^U by definition cannot be pre-empted in any team over T^{CBA} and M^U, our transition relation necessarily contains $((A_e, B_n, C_e, U_n), r^B, (A_e, B_r, C_e, U_n))$. Hence Kwaku, once in room B, can always begin reading file F. By not including $((A_n, B_e, C_e, U_n), e_{AB}, (A_e, B_n, C_e, U_n))$ in our transition relation we avoid that Kwaku can read F without ever having been granted read access. This leads to the question of the revocation of access rights.

As argued, (A_e, B_n, C_e, U_r) meaning that Kwaku is in room B with reading rights, will be a reachable state. Now imagine that while in this state Kwaku's reading rights are revoked by $\underline{m}(r)$. To which state should this action lead, i.e in what way do we handle revocation of access rights? We could opt for modeling *immediate revocation* or *delayed revocation*. The latter is what we have chosen to model first Thus our answer to the question above is to include $((A_e, B_n, C_e, U_r), \underline{m}(r), (A_e, B_n, C_e, U_n))$ in $\mathcal{T}$. The result is that Kwaku can pursue his activities in room B, but cannot re-enter the room once he has left it (unless his read access has been restored) He is thus still able to read (browse) F, but the moment he decides to re-open the file this fails Likewise, if Kwaku is writing F when his writing right is revoked, then he can continue editing (typing in) F, but he cannot re-enter room C as long as his write access right has not been restored. On this side of the revocation spectrum, the user can thus continue his current activity even when his rights have been revoked He can do so until he wants to restart this activity, at which moment an authorization check is done to decide if he has the right to restart this activity. In some applications, this may be an intolerable delay.

Immediate revocation, on the other hand, means the following. If a user is reading when his or her reading right is revoked, then the file immediately disappears from view, while if a user is writing when his or her writing right is revoked, then the edit is interrupted and writing is terminated in the middle of the current activity. In some applications, this is overly disruptive and unfriendly. If we would want to incorporate immediate revocation into our example we would have to adapt our distribution of actions a bit. As said before, since r^B is an internal action we cannot disallow action r^B to take place after $((A_e, B_n, C_e, U_r), \underline{m}(r), (A_e, B_n, C_e, U_n))$ has revoked Kwaku's reading rights. If we instead choose r^B to be an external action, we are given the freedom not to include $((A_e, B_n, C_e, U_n), r^B, (A_e, B_r, C_e, U_n))$ in our transition relation. The result is that as long as Kwaku is not being granted read access by action $m(r)$, the only way left to proceed for Kwaku in state (A_e, B_n, C_e, U_n) is to exit room B by $((A_e, B_n, C_e, U_n), e_{BA}, (A_n, B_e, C_e, U_n))$. Modeling immediate revocation thus requires that actions such as r^B are visible, since in that way we can choose them not to be enabled in certain states. Immediate revocation also implies that we still want Kwaku to be able to stop reading and leave state (A_e, B_r, C_e, U_n) by $((A_e, B_r, C_e, U_n), \underline{r}^B, (A_e, B_n, C_e, U_n))$. Action $\underline{r}^B$ can thus remain internal.

This finishes the description of part of a team automaton $\mathcal{T}$ over T^{CBA} and M^U In Figure 5 the full reachable part of $\mathcal{T}$ (for delayed revocation) is depicted

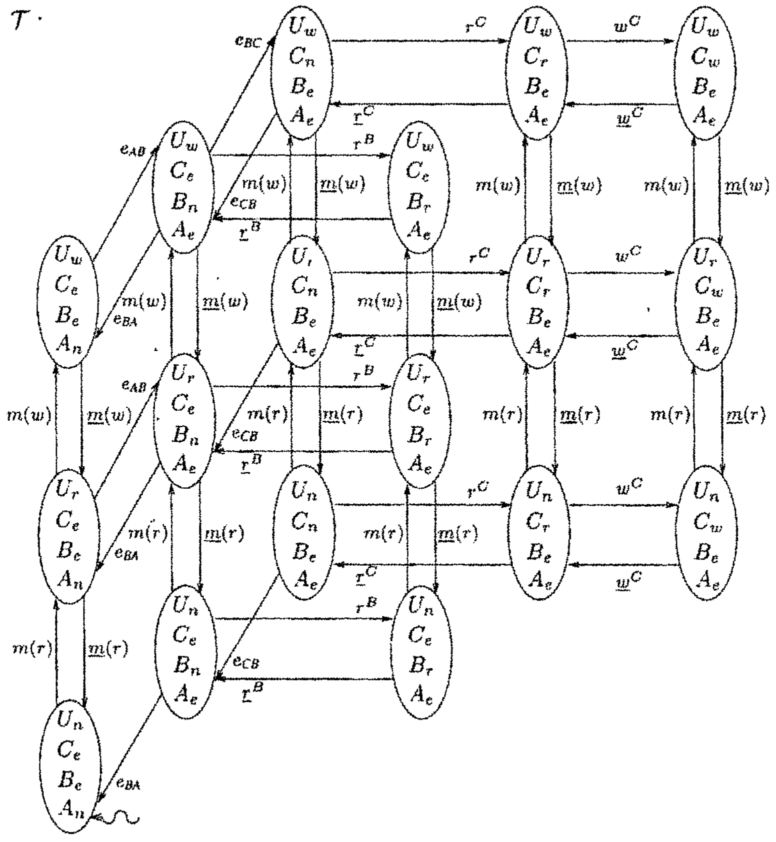

Figure 5 Team automaton $\mathcal{T}$ over T^{CBA} and M^U.

Note that team automata as dicussed here are used to model logical design issues. An action can take place provided (local) preconditions hold, and affects only states of those components involved in that action. Hence at this level there is no notion of time and no means are provided to give one action priority over another. A result of the lack of a notion of time is, e.g., that nothing can be said about how long it takes before Kwaku has left reading room B after his reading access right has been revoked. However, time and priorities may be added to the basic model as extra features.

Again, $\mathcal{T}$ is not the unique team automaton over T^{CBA} and M^U, but it is a team automaton one obtains by choosing a specific transition relation with a specific protocol in mind. In (ter Beek et al., 1999) certain fixed strategies for choosing transition relations in a predetermined way are described, which lead to uniquely defined team automata. One of these fixed strategies prescribes one to include for all actions a, all and only transitions on a in which all component automata participate that have a as one of their actions This leaves no choice for the transition relation, and thus leads to a unique team automaton. Constructing the transition relation according to this particular strategy is very natural and often presupposed implicitly in the literature. Note that the freedom of the team automata model to choose transition relations offers the flexibility to distinguish even the smallest nuances in the meaning of one's model. Leaving the set of transitions of a team automaton as a modeling choice is perhaps the most important feature of team automata.

Another interesting feature of the framework is shown by the following application of a result proved in Section 4 of (ter Beek et al , 1999) to our example. In whatever order one chooses to construct a team automaton over the component automata M^U, M^C, M^B, and M^A, it will always be possible to construct the team $\mathcal{T}$ discussed above. This means that instead of first constructing $\mathcal{T}^{CBA}$ over M^C, M^B, and M^A, and then adding M^U, we could just as well have constructed what we call an *iterated team* in (ter Beek et al , 1999) by, e g , starting from the user automaton M^U and adding successively the component automata M^C, M^B, and M^A modeling the access rights that can be exercised. Moreover, independent of the way the team over M^U, M^C, M^B, and M^A was constructed, more components can be added. As an example, suppose that Kwaku has other interests than the file F Hence imagine an automaton $\mathcal{T}^{NBA}$ in which he can transition into a state in which he plays some basketball Then we may construct a team over the team automaton $\mathcal{T}$ just described and the automaton $\mathcal{T}^{NBA}$ modeling when Kwaku is entitled — or perhaps even forced — to have a break (which is of some importance in these times of RSI). In general, new components can be added to a given team automaton at any moment of time, without affecting the possibilities of any new additions. The team automata framework thus scores high on scalability. In the next section we will come back to this.

Meta Access Control

In the previous sections we have seen how team automata can be used to describe the control of a user's access to a file depending on his or her rights. In this section we further elaborate on the granting and revoking of access rights and we consider *meta access control*. This means that privileges such as granting and revoking of rights can themselves be granted and revoked. The complicated (recursive) situations that may arise in this fashion depend on the chosen (meta) access control policy and we demonstrate how they can unambiguously and concisely be defined in terms of team automata.

Figure 6 shows an automaton M^0 that models a building with three levels — A, B, and C — corresponding to null access, read access, and full access, respectively.

This automaton shows the same access structure as the three rooms of Figure 2. Now, however, the status of the user directly determines the level he or she operates on and the granting and revoking of access rights is identified with changing levels. This differs from the previous example where the status of the user only determined his or her rights to enter a room.

M^0 :

$$\begin{tikzcd}
C_n \arrow[rr, bend left, "r"] \arrow[d, shift left, "\underline{m}(w)"] & & C_r \arrow[ll, bend left, "\underline{r}"] \arrow[rr, bend left, "w"] \arrow[d, shift left, "\underline{m}(w)"] & & C_w \arrow[ll, bend left, "\underline{w}"] \arrow[d, shift left, "\underline{m}(w)"] \\
B_n \arrow[u, shift left, "m(w)"] \arrow[rr, bend left, "r"] \arrow[d, shift left, "\underline{m}(r)"] & & B_r \arrow[ll, bend left, "\underline{r}"] \arrow[u, shift left, "m(w)"] \arrow[d, shift left, "\underline{m}(r)"] & & B_w \arrow[ll, "\underline{w}"] \arrow[u, shift left, "m(w)"] \arrow[d, shift left, "\underline{m}(r)"] \\
\arrow[r, squiggly] & A_n \arrow[u, shift left, "m(r)"] & A_r \arrow[l, "\underline{r}"] \arrow[u, shift left, "m(r)"] & & A_w \arrow[ll, "\underline{w}"] \arrow[u, shift left, "m(r)"]
\end{tikzcd}$$

Figure 6. Automaton M^0 the access building

Consequently, in M^0 the user moves in two dimensions: vertically between levels A, B, and C — indicating the dynamic change in access rights Kwaku has for F — and horizontally between the states "null", "reading", and "writing" — indicating the current activities of Kwaku with respect to F. Notice that in M^0, e.g , the state B_w meaning that Kwaku is writing while having read access but no write access, can only be reached from C_w by an action $\underline{m}(w)$ or from A_w by an action $m(r)$. Hence this state B_w can be entered only when Kwaku is writing while his status changes. There is no transition to B_w at level B A similar remark holds for states A_r and A_w, which can be entered only from level B by the read access revocation action $\underline{m}(r)$. States such as A_r, A_w, and B_w are called *irregular states* because they are not reachable at their own level.

To model meta access control, we assume the existence of a system administrator, Abena, who can change Kwaku's rights. Hence Abena has the right to grant and revoke access by Kwaku to F. For this reason we have chosen all actions of granting and revoking access rights in M^0 to be input actions, while all actions of reading and writing are output actions. The right to grant and revoke are legitimate rights, but they are not directly applied to F. They are in fact meta operations — hence $m(r)$ and $m(w)$ — and the rights to apply these meta operations are meta rights. Similarly, if there is a creator, Kwesi, who can allow (and disallow) Abena to grant and revoke, then Kwesi has meta meta rights. Kwesi has the meta meta right to grant and revoke Abena's meta rights to grant and revoke Kwaku's access rights to F. A typical action of Kwesi is $\underline{m}^2(w)$, which revokes Abena's right to grant and revoke write access to Kwaku

The notion of meta clearly extends to arbitrary layers. An example of such a multi-layered structure of meta can be seen in the journal refereeing process. The creator of a document may delegate publication responsibilities to co-authors who may select a journal and grant $m^2(r)$ rights to the editor-in-chief. The editor-in-chief may grant $m(r)$ rights to assistant editors who can then grant and revoke read access to reviewers. An interesting question now arises as to the effect of revocation. should revocation of a meta right also revoke the rights that were passed on to others? This is the issue of *shallow revocation* versus *deep revocation*. Shallow revocation means that a revoke action does not revoke any of the rights that were previously passed on to others, whereas deep revocation means that a revoke action does revoke all rights previously passed on. Team automata can be used to model shallow, deep, or even hybrid revocation. Shallow revocation is often the easiest to model, whereas deep revocation is known as a big challenge to model and implement (Dewan et al., 1998). We now show how deep revocation can be modeled using team automata.

Figure 7 shows an automaton capturing one layer (layer k) of a multi-layer meta access specification for our example of read and write access. We have already seen layer 0, viz. automaton M^0. For each value of $k \geq 1$ there are corresponding automata that are directly related to layer k (viz. M^{k-1} at layer $k-1$ and M^{k+1} at layer $k+1$). For each such automaton M^k, the horizontal actions $m^k(r)$, $\underline{m}^k(r)$, $m^k(w)$, and $\underline{m}^k(w)$ are output actions, whereas the vertical actions $m^{k+1}(r)$, $\underline{m}^{k+1}(r)$, $m^{k+1}(w)$, and $\underline{m}^{k+1}(w)$ are input actions. For $k=0$ we identify r with $m^0(r)$, $\underline{r}$ with $\underline{m}^0(r)$, w with $m^0(w)$, and $\underline{w}$ with $\underline{m}^0(w)$. Similarly, $m(r) = m^1(r)$, $\underline{m}(r) = \underline{m}^1(r)$, $m(w) = m^1(w)$, and $\underline{m}(w) = \underline{m}^1(w)$.

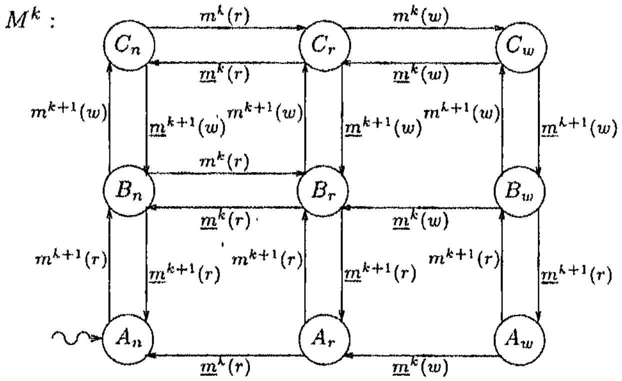

Figure 7 Automaton M^k meta access at layer k

We can now define a multi-layered structure by recursively composing a team automaton over M^0, M^1, . . ., and M^n, for some $n \geq k$. Note that this is a composable system. As mentioned before we can also build this team automaton in an iterated way starting from, e.g., a team over any two automata M^k and M^{k+1}. In Figure 8, the reachable part of a team automaton T^k_{k-1} over M^{k-1} and M^k, representing layer $k-1$ and layer k of this layered structure, is depicted. The transition relation of this team T^k_{k-1} is chosen with the modeling of deep revocation in mind. Finally, note that in Figure 8 we have added superscripts to distinguish the states in M^k from the states

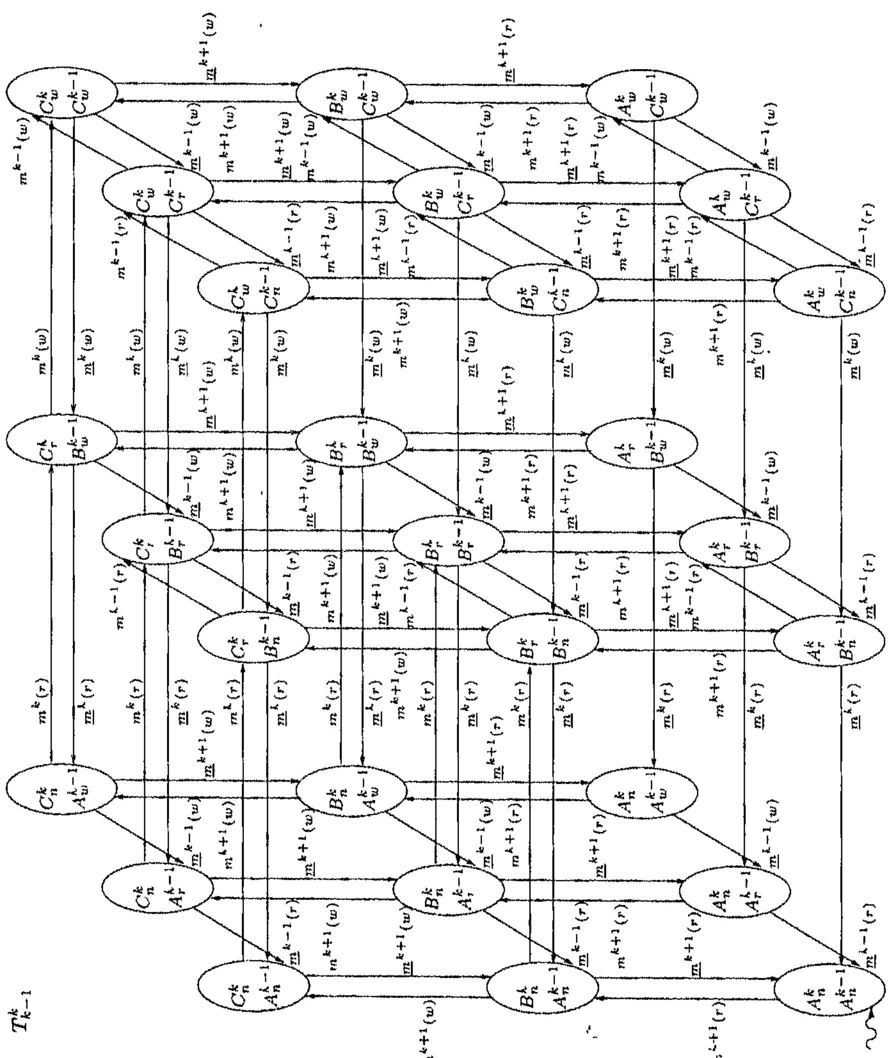
Figure 8 Team automaton T^k_{k-1} over M^{k-1} and M^k

in M^{k-1}, e.g., state B_r of M^k from state B_r of M^{k-1}

In our example, M^2 represents the actions of the supervisor Kwesi and M^1 those of Abena. Now consider Kwesi in state B_r^2. Then Figure 8 tells us that Abena must be in one of the three states B_n^1, B_r^1, or B_w^1. Assume that Kwesi reached this state B_r^2 by performing action $m^2(r)$ from B_n^2, while Abena was in state A_n^1 having no rights to grant and revoke reading rights. Action $m^2(r)$ is an output action of M^2 and an input action of M^1, and our transition relation forces M^1 to transition from A_n^1 to B_n^1 The interpretation is that Kwesi granted Abena the right to do read grants and revokes (to user Kwaku for file F).

Similarly, automaton M^k can revoke the right to grant and to revoke read access from M^{k-1} at any time by performing output action $\underline{m}^k(r)$, and thus forcing M^{k-1} to perform this action — this time as an input action — as well. Continuing our example, this means that while in state B_r^2, Kwesi's read granting right may be revoked by action $\underline{m}^3(r)$ at any time. If this happens, Kwesi is forced into the irregular state A_r^2, which has only one possible output action, viz. $\underline{m}^2(r)$, leading to A_n^2. Whenever that action $\underline{m}^2(r)$ occurs it revokes Abena's right to change Kwaku's read access.

We thus observe two general rules of activity in such a team automaton over M^0, $M^1, \ldots$, and M^n with each automaton of the form depicted in Figure 7. First, when a master automaton M^k where $1 \leq k \leq n$, transitions right (grant) or left (revoke), then the slave automaton M^{k-1} must transition upward (gaining some access right) or downward (losing some access right). Second, the slave M^{k-1} may be forced to transition downward into an irregular state, in which case it will eventually transition to the left M^{k-1} is itself a master and thus this transition to the left again forces a downward transition of M^{k-2}, and so on until M^0 on layer 0 Hence, as promised, we indeed model deep revocation

Conclusion

In this document we have demonstrated by means of examples how team automata can be used for modeling access control mechanisms presented through the metaphor of spatial access. The combination of the formal framework of team automata and the spatial access metaphor leads to a powerful abstraction well suited for a precise description of (at least some of the) key issues of access control The team automata framework supports the design of distributed systems and protocols, by making explicit the role of actions and the choice of transitions governing the coordination (e.g., in the form of peer-to-peer or master-slave synchronizations, or combinations thereof). Moreover, the formal setup and the possibility of a modular design provide analytic tools for the verification of desired properties of complex systems. Team automata are thus a fitting companion to the virtual spaces metaphor used in virtual reality systems that supports notions of rooms and buildings. Each space is represented by a component automaton, dynamic access changes are represented by joint external actions, while resource accesses within a space can be represented by internal actions. For reasons of readability, we have chosen for a presentation by examples without definitions and proofs. Obviously there are numerous other possible examples as well as variations of the examples we have considered.

For one, the assumption that write access can only be granted if read access has been granted can easily be dropped. Similarly, grant and revoke rights can be coupled more loosely. Read and write operations are specified here at the file level, but could also have been specified at the page level, object level, or record level, to name but a few. This might mean that delayed revocation is precisely the right choice. At the file level, the r and $\underline{r}$ actions might be seen at the user interface as open and close file. The w and $\underline{w}$ actions might be edit and save operations. When dealing with a transaction system, combinations of these operations might correspond to begin transaction and end transaction.

The team automata framework handles group decision making well and therefore allows convenient implementations of *distributed access control*. Distributed access control means that the supervisory work of granting and revoking access rights is administered by multiple agents. Thus Kwaku could have two administrative supervisors who must agree on any change of access rights. This can be modeled as an action of two masters and one slave: the actions would be output for both supervisors, requiring both to participate, and input for the slave. Alternatively, by including transitions with one supervisor being inactive, we can model the case of approval being required by either one of the two supervisors. Hybrids between pure master-slave and pure peer-to-peer are easy to define, and useful. All these variations are due to the fact that the choice of a transition relation is the crucial modeling issue of the team automata framework.

Note that team automata model the logical architecture of a design. They abstract from concrete data, configurations, and actions, and only describe behavior in terms of a state-action diagram (structure), the role of actions (input, output, or internal), and synchronizations. It is not feasible (nor necessary) to have a distinct automaton for each individual, and for each file in an organization. In many situations, categories and roles are used rather than individuals. Any implementation would have the team automaton as a class entity, and an activation record for each person, containing their current state. Similarly, by keeping a status of the files one can model the criterion "only one person can write a file at a time, but many readers is OK". The model cast in the spirit of automata depicting roles rather than individuals becomes much more useful and general, and avoids some notational problems of exponential growth. Note that because of the product construction, a state space explosion lurks. However, the resulting automata are not difficult to process. The iterative approach to composition (forming teams with teams as components) forms an automatic and mechanizable abbreviation methodology (van der Aalst et al., 2000). In general, however, the state space explosion problem itself cannot be avoided when dealing with systems with many components that can interact or can assume many different values. See (Clarke et al., 1999) for a presentation of techniques and tools for dealing with this problem when verifying large systems. (Müller, 1998) demonstrates how to deal with the computer-assisted verification of embedded systems described as I/O automata.

As observed earlier, time and priorities are not incorporated in neither the spatial access metaphor nor the team automata model as discussed here. However, similar to the Petri net model — which is also based on local state changes — one may consider to extend team automata with time and priorities (see, e.g., (Ajmone Marson et al., 1995), which focuses on performance analysis). When time and/or priorities are part of access control this would allow the designer to control the sojourn times in the local states and to control the resolution of conflicting actions.

Using team automata for modeling (spatial) access control forces one to make explicit and unambiguous design choices and at the same time provides the possibility of mathematically precise analysis tools for proving crucial design properties, without first having to implement one's design.

To conclude we stress that (spatial) access control is only one of many CSCW concerns that can be addressed via team automata. The higher goal of this document is to demonstrate the applicability of a formal framework that was conceived specifically as a model for the specification and analysis of CSCW systems. We believe that this may be a significant step toward a better understanding of the ways in which people and systems cooperate and collaborate.

Acknowledgments

We are grateful to the anonymous referees for their suggestions to improve an earlier version of this document.

References

van der Aalst, W., Barthelmess, P., and Ellis, C.A. (2000): 'Workflow Modeling using Proclets'. Technical Report CU-CS-900-00, Computer Science Department, University of Colorado.

Ajmone Marson, M., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G (1995): *Modelling with generalized stochastic Petri nets*, John Wiley & Sons, Chichester

Attie, P.C., Francez, N., and Grumberg, O. (1990): 'Fairness and hyperfairness in multi-party interactions', in *Proceedings of the POPL'90 ACM Symposium on Principles of Programming Languages, San Francisco, California*, ACM Press, pp. 292-305.

ter Beek, M H., Ellis, C.A., Kleijn, J., and Rozenberg, G (1999): 'Synchronizations in Team Automata for Groupware Systems'. Technical Report TR-99-12, Leiden Institute of Advanced Computer Science, Universiteit Leiden.

Brookes, S D , Hoare, C.A.R., and Roscoe, A.W. (1984): 'A theory of communicating sequential processes', *Journal of the ACM*, vol 31, no. 3, pp. 560-599.

Bullock, A. and Benford, S. (1997): 'Access Control in Virtual Environments', in D Thalmann, S. Feiner, and G. Singh (eds.): *Proceedings of the VRST'97 ACM Symposium on Virtual Reality Software and Technology, Lausanne, Switzerland*, ACM Press, pp. 29-35.

Bullock, A. (1998). *SPACE: Spatial Access Control in Collaborative Virtual Environments*. Ph.D. thesis. Department of Computer Science, University of Nottingham

Bullock, A. and Benford, S. (1999): 'An access control framework for multi-user collaborative environments', in *Proceedings of the GROUP'99 International ACM SIGGROUP Conference on Supporting Group Work, Phoenix, Arizona*, ACM Press, pp. 140-149.

Clarke Jr., E M., Grumberg, O., and Peled, D.A (1999): *Model Checking*, MIT Press, Cambridge, Massachusetts.

Dewan, P and Shen, H (1998): 'Flexible Meta Access-Control for Collaborative Applications', in E. Churchill, D. Snowdon, and G Golovchinsky (eds). *Proceedings of the CSCW'98 ACM Conference on Computer Supported Cooperative Work, Seattle, Washington*, ACM Press, pp. 247-256.

Ellis, C.A. (1997): 'Team Automata for Groupware Systems', in J. Clifford, B Lindsday and D Maier (eds.): *Proceedings of the GROUP'97 International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge, Phoenix, Arizona*, ACM Press, pp. 415-424.

Evangelist, M , Shen, V.Y., Forman, I R., and Graf, M. (1988): 'Using Raddle to design distributed systems', in *Proceedings of the ICSE'88 International Conference on Software Engineering, Singapore*, IEEE Computer Society Press, pp. 102-111.

Keesmaat, N.W. (1996): *Vector Controlled Concurrent Systems* Ph.D. thesis, Leiden University.

Lynch, N.A. (1996): *Distributed Algorithms*, Morgan Kaufmann Publishers, San Mateo, California

Müller, O (1998): *A Verification Environment for I/O Automata Based on Formalized Meta-Theory* Ph.D. thesis, Technische Universität München

Nutt, G.J. (1997) *Operating Systems: A Modern Perspective*, Addison-Wesley Publishers, Reading, Massachusetts.

Reisig, W., and Rozenberg, G. (eds) (1998): *Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science*, vol. 1491, Springer-Verlag, Berlin.

Rodden, T. (1996): 'Populating the Application: A Model of Awareness for Cooperative Applications', in M. Ackerman (ed.): *Proceedings of the CSCW'96 ACM Conference on Computer Supported Cooperative Work, Boston, Massachusetts*, ACM Press, pp. 87-96.

Shen, H. and Dewan, P. (1992): 'Access Control for Collaborative Environments', in J. Turner and R Kraut (eds.): *Proceedings of the CSCW'92 ACM Conference on Computer Supported Cooperative Work, Toronto, Canada*, ACM Press, pp. 51-58

Sikkel, K. (1997): 'A Group-based Authorization Model for Cooperative Systems', in J. Hughes, W. Prinz, T. Rodden, and K. Schmidt (eds.): *Proceedings of the ECSCW'97 European conference on Computer Supported Cooperative Work, Lancaster, UK*, Kluwer Academic Publishers, pp. 345-360.