Enhanced Security Management for Flexible and Dynamic Cooperative Environments

Jonas Schulte*§, Ingo Döpke*, Reinhard Keil*‡, Konrad Stark†, and Johann Eder‖
*University of Paderborn, Heinz-Nixdorf-Institute, 33102 Paderborn, Germany
Email: §schulte@uni-paderborn.de ‡Reinhard.Keil@uni-paderborn.de
†University of Vienna, Department of Knowledge and Business Engineering, 1010 Vienna, Austria
Email: konrad.stark@univie.ac.at
‖University of Klagenfurt, Department of Informatics Systems, 9020 Klagenfurt, Austria
Email: eder@isys.uni-klu.ac.at

Abstract— **This paper deals with the challenge to create an authorization and authentication infrastructure for virtual knowledge spaces. Virtual knowledge spaces are a concept to build up flexible and adjustable environments for cooperative work and learning processes. When developing authorization concepts for virtual knowledge spaces, different boundary conditions have to be considered. Basically, we identify the specific requirements for the creation of coherent and intuitive rules to reduce the administrative complexity to a minimum and prevent mistakes. Furthermore we turn attention to flexibility of the authorization infrastructure as well as performance issues. The rights management described in this paper ought to fulfill these requirements.**

I. Introduction

We are living in a changing information society. The way of handling electronic information changed fundamentally during the past years. The original internet as a platform of total freedom without supervision mutated to a place which is characterized by monitoring and observation [1], [2]. These movements are critical with respect to the privacy protection of internet users. Various social communities lack sufficient settings to specify detailed access rights. Especially for applications in the area of *Computer Supported Cooperative Work (CSCW)* it is essential to ensure data security. This aspect becomes even more important if the CSCW application is used for safety critical business areas, e.g. medical research with sensitive patient data [3].

In this paper we present an authorization infrastructure with a novel rights management system applied to cooperative work environments. Furthermore our development satisfies the requirements of virtual knowledge spaces (see section II). The presented authorization infrastructure is not limited to virtual knowledge spaces and can be adopted to different fields of application.

The paper is structured as follows. In section II we introduce virtual knowledge spaces by introducing the common structure, motivate the concept of multiple views, and present scenarios to exemplify the power and the flexibility. Section III describes the basic data structure of the developed authorization infrastructure and draws a distinction between the different types of rights. Afterwards we explain in section IV the hierarchy of rights and the concept of inheritance. Section V explains all functions of the rights management in detail. To improve the performance of the authorization infrastructure we developed some methods that enable fast security checking. Furthermore we give different scenarios in section VII to demonstrate the functionality of the rights management functions as well as the authorization functions.

II. Knowledge Spaces – A Concept for Dynamic and Flexible Cooperative Work

Virtual knowledge spaces are a key concept to establish dynamic and flexible cooperative work and learning environments. It allows unrestricted structuring of data and provides means for generating multiple views on the same data collection (see section II-B). Thus, it is possible to adjust the environment exactly to the users' needs, without having redundancy of the underlying data. The concept of virtual knowledge spaces has been proven for more than several years and in various fields of application [4], [5]. The following section II-A describes the class hierarchy for setting up virtual knowledge spaces. After that we show how to create different views of a data collection (section II-B).

A. Common Structure of Knowledge Spaces

Knowledge spaces are our concept for the representation and the structuring of information within a cooperative work process. This the concept has been applied to various fields of application and is well-proven due to almost ten years experience of software development in the scope of CSCW. [6]

The basic object structure of virtual knowledge spaces is quite easy. We distinguish between the following objects: `Room`[1], `Container`, `Document`, `Attribute`, `Link`, `Group` and `User`. A `Room` is the representation of a real room, whereas `Container` can be compared with a folder. The intention behind the `Room`-object is having an object, that stands for a place where uses can meet each other and cooperate with each other. Furthermore spaces are intended to assure awareness. Awareness is indispensable for successful CSCW applications [7], [8]. On the other hand

[1] In the object model a `Room` stands for a virtual knowledge space.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

the `Container`-object is a lightweight construct to organize information and correlate data. Objects of the type `Document` are used as a wrapper around the real content (e.g. a picture, a word document, ...). There is a fundamental difference between on the one hand storing content that is linked with additional information and on the other hand an object that consists of the content as well as the additional information. The first option relates different data to each other, whereas the second and preferred option builds up a *combined data object*. For further processing a combined data object has different advantages. For instance, when creating versions of an object it is a lot easier to generate a coherent version object of a combined data object instead of creating equivalent versions of all related objects that match the combined data object. Our model allows free attributes at any type of object. Thereby the model is highly customizable to the requirements of the current application area. `Group`-, `User`- and `Link`-objects are self-explanatory. It is mentionable, that we use `Group`-objects not only for grouping users, but also for illustrating roles.

We developed a framework named WasabiBeans that implements the concept of virtual knowledge spaces. WasabiBeans is designed as service oriented architecture and implemented in JavaEE with EJB 3.0 running on JBoss AS [9]. For further information about the WasabiBeans framework and its architecture please refer to [10].

B. Multiple Views on a Data Collection

Different applications have different types of data representation. Therefore usually each application comes with its own repository containing the related data. Our understanding of handling information in cooperative work environments is to work on his or her own data collection and make these data available in different fields of application. Several reasons argue for the usage of one data collection instead of creating data redundancies. In order to avoid problems as updating different data collections or the distribution of changes among various repositories, the usage of a common data space is essential. When setting up an uniform data collection, it is important to provide different interfaces. WasabiBeans intends to be a flexible and adjustable framework and therefore provides various interfaces to access the data.

Unique and creative concepts, e.g. the pyramid discussion for decision-making processes desire totally different views and representation types [11]. But even the presentation of miscellaneous file formats benefit from different views. Image files might be displayed as a picture gallery, whereas documents are listed alphabetically. There are boundless possibilities to display a data collection or a set of a data collection. It is obvious, that different views are meaningful. Making one data collection available in different applications requires a sophisticated rights management. In section III we define the basic elements of the rights management.

C. Example of Usage

The advantages of virtual knowledge spaces have special effect when using them in cooperative environments, that are characterized by dynamic interaction among the participants and changing circumstances. An example of intensive collaboration are joint research projects among universities, research facilities, and companies. This type of collaboration requires more than simple document management. Particularly large joint research projects compose of different sub-projects and sub-tasks. Since not all participants work on the same task, the creation of sub-groups is necessary. For each sub-group one knowledge space can be created as a meeting point and work place for the collaboration. Hence virtual knowledge spaces support awareness among users that are present in the same space, the collaboration is oriented towards the important participants. Nevertheless the cooperation is not limited to members of sub-groups, because sub-group members can also join a higher parent group and enter this group space for inter-group collaboration.

III. Basic Elements

In this section the basic data structure of the authorization infrastructure is described. The most important data element is the right item. This item stands for the different types of right assignments that can be made by an user. A right item features several properties and is used by the authorization functions, introduced in section V, as described below.

A. Permissions

Unlike rights, which exhibit a complete rule of the form Who - What - Whom, a permission is an atomar unit. It does not give any evidence about the affiliation, but only classifies the kind of a specific right. Accordingly a permission should be understood as a property of a right (see section III-B). To implement a flexible right management the WasabiBeans framework comes with its own set of permissions. In total we identified seven permissions (`VIEW`, `READ`, `EXECUTE`, `INSERT`, `WRITE`, `COMMENT`, `GRANT`), that might be extended for specific needs. The meaning of each permission specified in the `WasabiPermission` class may vary depending on the objects the permission belongs to.

1) VIEW:

- **`WasabiUser`**: Ability to see a user.
- **`WasabiGroup`**: Ability to see a group.
- **`WasabiDocument`**: Ability to see a document.
- **`WasabiLocation`**[2]: Ability to see a space or a container.
- **`WasabiAttribute`**: Ability to see an attribute.

An example for the visibility of users can be given by the following scenario: All users of a space are visible, except the admins. This is meaningful, since usually the users shouldn't get in direct contact with the server admins. Therefore the admins are not listed in the list of visible persons. By using

[2]WasabiLocation is an abstract object combining the WasabiContainer and WasabiRoom.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

the VIEW right the cognition of any object can be customized. Another use case for the VIEW right is the scenario that a user wants to make itself invisible, in order not to be contacted by others at the moment.

2) READ:

- **`WasabiUser`**: Access to all parameters of a user.
- **`WasabiGroup`**: Access to all parameters of a group.
- **`WasabiDocument`**: Privilege to read a document.
- **`WasabiLocation`**: Privilege to enter a space or a container.
- **`WasabiAttribute`**: Privilege to read the value of an attribute.

3) EXECUTE:

- **`WasabiUser`**: Not defined.
- **`WasabiGroup`**: Not defined.
- **`WasabiDocument`**: Privilege to execute a file (e.g. a script).
- **`WasabiLocation`**: Not defined.
- **`WasabiAttribute`**: Not defined.

4) INSERT:

- **`WasabiUser`**: Privilege to insert this user to a group or to deprive him from it.
- **`WasabiGroup`**: Privilege to give or deprive other users the group membership.
- **`WasabiDocument`**: Not defined.
- **`WasabiLocation`**: Privilege to insert objects into the space or the container.
- **`WasabiAttribute`**: Not defined.

To insert a user to a specific group, the executer must have an INSERT on the group and as well on the user. However, if a user should be removed from a group, a GRANT on the group is enough. The reason for this consists of the fact that it must always be allowed to remove a user from a group which is a property of the executer.

5) WRITE:

- **`WasabiUser`**: Privilege to change and delete all parameters of an user. It covers the INSERT right too.
- **`WasabiGroup`**: Privilege to change and delete all parameters of aa group, except the modification of group–subgroup relation. It covers the INSERT right too.
- **`WasabiDocument`**: Privilege to rename, delete, and change a document.
- **`WasabiLocation`**: Privilege to rename, delete, and insert objects into a space or a container. It covers the INSERT right too.
- **`WasabiAttribute`**: Privilege to change the value or delete the attribute.

The WRITE permission covers the same as the INSERT permission and extends it with additional features.

6) COMMENT:

- **`WasabiUser`**: Privilege to annotate this user with tags.
- **`WasabiGroup`**: Privilege to annotate this group with tags.
- **`WasabiDocument`**: Privilege to annotate this document with tags.
- **`WasabiLocation`**: Privilege to annotate this space or document with tags.
- **`WasabiAttribute`**: Privilege to annotate this attribute with tags.

In fact tags could also be realized by using attributes, however we think tags are a very important and widely used method to add catchwords. Due to the frequent usage of tags we implemented tags as strings, that are easy to use and with increased performance compared to the free attributes.

7) GRANT:

- **`WasabiUser`**: Privilege to grant permissions on this user.
- **`WasabiGroup`**: Privilege to grant permissions on this group. In addition it is needed to change the group–subgroup relation. Therefore the executor must have GRANT permission on both groups (subgroup and superordinated group). The GRANT permission is also required to delete a group. In addition, it allows to exclude users from a group.
- **`WasabiDocument`**: Privilege to grant permissions on this document.
- **`WasabiLocation`**: Privilege to grant permissions on this space or container.
- **`WasabiAttribute`**: Privilege to grant permissions on this attribute.

GRANT allows the declaration of new access rights as well as to change and delete them. A user who has GRANT permission on a specific object can be called the object's owner. As a result, the same object may be owned by (also called *"administrated by"*) more than one person.

B. Rights and their properties

As shown above, there exist seven different kinds of permissions in WasabiBeans. A permission alone does not have any impact, instead it is a property of a complete right. Because of the fact WasabiBeans uses discretionary access control, every right should have at least the following properties:

- allocation to a specific subject
- kind of permission
- allocation to a specific object[3]

This basic structure may be extended. As an additional property, we can consider the following one:

- type of right (allowance or forbiddance)

In our concept a forbiddance is defined to be more powerful than an allowance. Resulting from this a forbiddance may overlap an allowance, since more than one relevant right for a specific relation of *Subject–Permission–Object* can be defined.

[3]In WasabiBeans, the rights are stored directly in the objects they are allocated to.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

This may occur when setting group rights. In this manner, the property

- status of hierarchy

is also important. In case the authorization function discovers more than one relevant right, the system has to decide about which right is preferred.

There is the rule that a forbiddance is preferred to an allowance when belonging to the same hierarchy level. Assuming the levels forbiddance and allowance constitute their own hierarchy, this implicates the effective number of hierarchy levels is twice compared to the number of explicit ones. It should be obvious, that without forbiddance the generation of different hierarchy levels is needless. Resulting a right may have three different conditions: It can be either an allowance, a forbiddance, or not specified.

As an additional property a time interval can be specified for rights in WasabiBeans. Section III-E deals with this issue and explains the arising consequences for the authorization infrastructure. Furthermore, there are different criteria that has to be interpreted for the validation of a right. They are described in detail below.

C. User Rights

An access right always refers to a specific WasabiUser object. It assesses an operation for which the user is authorised on the specific data object or for which he is forbidden. Because of the fact that a WasabiBeans environment may contain plenty of different users, it is not practicable to allocate each conceivable right as a user right. To ease the right allocation and to reduce the number of granted rights, there exist the group rights (see III-D).

D. Group Rights

Within its function, a group right is similar to a user right, with the difference that it is not granted to a specific user, but a whole group of users. Because of this fact, it is sufficient to be a member of the specific group or the corresponding subgroups to gain all associated allowances and forbiddances. If a WasabiUser is member of two different groups, with at least one group which grants an allowance, this allowance overlaps each forbiddance which results from the membership on another group (see IV-A).

By the concept of group rights it is possible to define special fields of responsibility. For example, a group can be created whose members may access on special parts of the object tree to perform well defined operations there. If we want to allocate some users for this field of responsibility, we only need to assign them to this group. A declaration of additional user rights is not necessary.

E. Time Rights

As a difference to the common infinite rights, a time based right is only valid for a certain interval. This means, it is similar to a common right which is expanded by a starting point and an ending point. These two points in time are expressed in the number of milliseconds, which are elapsed since the first january 1970. Refering to this, a time right gets in use if the actual computer time is inside this interval. In a regular case, interferences with other time rights are always avoided by the system. This means, if a new time right is granted, whose interval overlaps at least one of another relevant time right, this interval may be shortened. In an extrem case, it may also be splitted in two other time rights or totally annulated.

If we abstain from the concept of time rights, a temporary allowance or forbiddance must always be initiated by the administrator to the designated point of time and annulated at another point of time. Thereby, the administrative effort would be elevated and the flexibility of the system would be decreased.

1) Dealing with occlusions: As it was already mentioned in this chapter, by the initiation of new time rights, there may occur inconsistences to other time rights which already exist. These inconsistences are dissolved by preferring the new time right. This means, it may occlude existing rights. To ensure this, before the initiation the system must check every available time right which contains the same user or group name and the same allocated object. Now the interval of the new right becomes compared with the one of the old right. It is $N = (N_1, N_2)$ the interval of the new time right and $O = (O_1, O_2)$ the interval of the old time right. There is the condition $N_1 \leq N_2$ and $O_1 \leq O_2$. Now we can divide between the following cases (the above cases exclude the below cases, the system works in sequence during the check):

1) $N_2 < O_1$ or $N_1 > O_2 \rightarrow$ There are no difficulties because the two intervals do not interfere.
2) $N_1 \leq O_1$ and $N_2 \geq O_2 \rightarrow$ The old time right becomes replaced by the new one.
3) $N_1 \leq O_1$ and $N_2 < O_2 \rightarrow$ The old time right is shortened on $(N_2 + 1, O_2)$.
4) $N_1 > O_1$ and $N_2 < O_2 \rightarrow$ The old time right becomes splitted in two subrights with the intervals $(O_1, N_1 - 1)$ and $(N_2 + 1, O_2)$.
5) $N_1 > O_1$ and $N_2 \geq O_2 \rightarrow$ The old time right becomes shortened on $(O_1, N_1 - 1)$.

F. Unlimited Rights

An unlimited right has got the same function as a time right with the difference, that there is given no interval. Of course there is the possibility to abstain from unlimited rights and to generate each right as a time right. By this case, an unlimited right would be a time right with the highest possible interval. But such a decision leads to a problem: If we generate a new time right, an old, virtual unlimited right which collidates with it would be cutted in two subrights (see III-E1). From this, the following disadvantages result:

- The old, once unlimited right should be deleted $\rightarrow$ Now we have to delete two subrights.
- The new time right should be deleted $\rightarrow$ This results in a gap.

As a consequence, we generate two different layers for the differenciation of time rights and clearly unlimited rights. The

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

time based layer should be above the unlimited one, because it can be considered as more special. The figure 1 should illustrate this concept. Another advantage of it results from the fact that we always have the possiblity of a clearly abdication from time based rights.

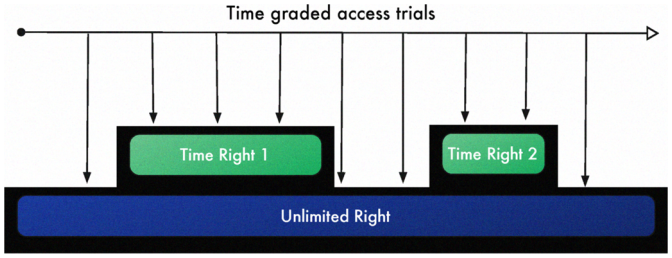

Fig. 1. The two time layers

G. Inherited Rights

The adjustment of data inside a tree structure suggests that access rights may be passed from a high into a junior instance, which can be considered as inheritance. If we abstain from this possibility, it is necessary to grant explicit rights for each single data object which increases the administrative effort. In addition to this, on ordinary scenarios we often have got the case, that all objects inside a junior structure have the same or at least very similar rights. The possibility of inheritance should support this.

Basically, we have got the principle that with two or more relevant rights within the same triple relationship (subject, permission, object), always the right which is nearer to the target object has got priority. This means, if there are for example two containers above a WasabiDocument, in which is in each case indicated a right for the same user, the right of the lower container is preferred. If there is even a collision of an inherited right with an explicit right, the inheritance may not occur. For time rights, there again is the principle that the rights with the shorter distance to the specific object may shorten, occlude or divide the intervals of the rights with the larger distance.

IV. General Rules

A. The hierarchy of rights

As ist was already mentioned in the last section, there exist three different criteria, which decide about the scope of a right:

1. Realm: User or group
2. Period of validity: Infinite or time based
3. Ancestry: Explicit or inherited

Altogether, we have $2^{Number\ of\ criteria} = 8$ different kinds of rights. If there is a check if a user may perform a specific action, there can be the problem that several rights are found, which differ in their kinds. For this case, it is necessary to build up a hierarchy, which allows a clear decision which right should be preferred. The graduation, which WasabiBeans uses is shown by figure 2.

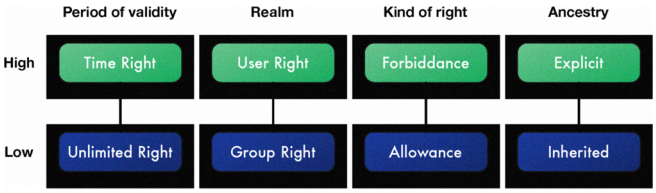

Fig. 2. Hierarchy of rights

As a consequence from this, user rights are always preferred before group rights. This has got several reasons: On the one hand we estimate from the fact, that in most cases there are only group rights and a user right is something special. On the other hand the administrator has got a back door to exclude some users from a group forbiddance without the need of removing them from the group. It can be said that this differentiation between user rights and group rights forms the only two explicit hierarchy levels of WasabiBeans. The other levels (explicit right vs inherited right, time based vs unlimited) are no more than rules to decide which rights are relevant.

As it was already mentioned in III-F, a time right may cover an unlimited right with the same triple relationship (subject, permission, object). For this reason it can be considered as a transaction: If the interval of the time right is reached, the unlimited right becomes deactivated and the time right gains the function of an unlimited right. After leaving the interval, the time right loses its function and the unlimited right recovers activation. This functionality of time rights can be concluded from figure 1. In a similar manner, the differentiation between explicit and inherited rights is handled: If we have got more than one right with the same triple relationship, we always take the right with the lowest distance to the target object. The subsection IV-B provides further information about this.

B. The concept of inheritance

As it was already mentioned in III-G, a right is always inherited from the superordinated instance. Because of our architecture model, the type of a WasabiObject decides to which other WasabiObject types its right may become inherited. There exist the following possibilities:

- WasabiRoom → WasabiRoom, WasabiContainer, WasabiDocument, WasabiAttribute
- WasabiContainer → WasabiContainer, WasabiDocument, WasabiAttribute
- WasabiDocument → WasabiAttribute
- WasabiGroup → WasabiGroup[4], WasabiAttribute
- WasabiUser → WasabiAttribute
- WasabiAttribute → WasabiAttribute

Up to this, there must be additional rules which decide if an inherited right becomes covered by an existing, explicit right:

[4]Has to be a subgroup

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

- Both rights have got an unlimited period of validity. This leads to a total covering (no inheritance).
- The right with the lower distance is unlimited, the other is time based. This also leads to a total covering of the inherited right, to support the transaction concept of time based rights (mentioned in III-G).
- Both rights are time based. In this case, the inherited right can be handled by the rules which are mentioned in III-E1.

A term which can be associated with the covering of right is the so called substructure.

Definition 1: A substructure always opens if a should be inherited right becomes covered totally or partially by an explicit right, which means there is a collision. It contains the property that it persists in most cases if there is a change in the object tree above.

V. Right Management Functions

A. Implementing the concept of inherited rights

To implement the concept of inheritance, we have to consider two different approaches:

1) If the authorization function has to decide about the allowance of a specific operation on a target WasabiObject, it may run up to the root of the object tree applying the inheritance rules to get all considerable rights.
2) The right declaration functions may propagate all should be inherited rights from each WasabiObject down to the bottom of the object tree. These lists of inherited rights may be used by the authorization function in a direct manner.

The architecture of WasabiBeans uses the second approach. The reason for this decision was because we emanate from the fact, that the declaration of rights is more rarely used than the check of them. For this reason, the additional effort of propagation may be tolerated if it speeds up the authorization.

B. Using Propagation

A basis for the use of propagation is given by two additional structures:

1) It must be possible to decide whether a right is explicit or inherited.
2) Each WasabiObject must contain a flag which shows if it is allowed to inherit the rights from the superordinated object.

Figure 3[5] shows an example for a structure with inheritance. As it should be obvious, the explicit right of the container 'Pictures' forms a substructure, because it blocks a should be inherited right from above.

There is also the demand that this extended structure contains a consistent state. A right structure is always in a consistent state, if the inherited rights directly result from the explicit rights by the use of the existing rules of inheritance and occlusion. An example for an inconsistent state is given by figure 4.

[5]Right marked with an (e) are explicit rights, those marked with an (i) are inherited rights.

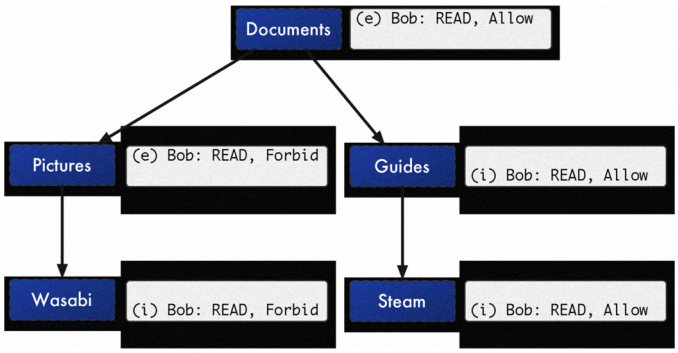

Fig. 3. An example for inheritance

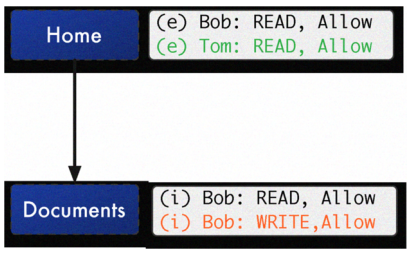

Fig. 4. An example for two inconsistences

According to this, the used propagation functions must always ensure consistency, otherwise they don't work correctly. Now we have got the question which kinds of propagation we need. The following scenarios have to be considered:

1) A new right is granted → It must be propagated down the subtree.
2) An old right is changed → The change must be propagated down the subtree.
3) An old right is removed → All inherited rights which result from this old right must also be removed.
4) A new object is created → It must inherit the rights from its location.
5) An existing object is moved → The object and its subtree must lose the inherited rights from the old location and gain the rights of the new one.
6) The inheritance of an object is deactivated → The inherited rights from its location must be removed.
7) The inheritance of an object is activated → The rights from its location must be gained as inherited rights.

Altogether, we have got four different propagation functions which can handle all these scenarios:

1. Inherit rights: This function can be used for scenario 3 (if the removed right was a substructure to a right from above, this right from above must be propagated again), 4, 5 (gaining the rights of the new location) and 7.
2. Disinherit rights: This function can be used for scenario 5 (removing the inherited rights of the old location) and 6.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

3. Grant rights: This function can be used for scenario 1 and 2 (grant the changed right again to overwrite the old version).
4. Remove rights: This function can be used for scenario 3.

According to this, we also use an additional propagation function:

5. Reset rights: Remove all explicit rights in the subtree, but not in its root.

With the use of Reset, it is possible to clear up a subtree from all its substructures and to grant that only the rights of its root have got influence down below.

C. Basic structures for propagation

All of the propagation functions which were mentioned in V-B have got the similarity that they must run through a subtree of WasabiObjects. This functionality can be granted by a single basic algorithm, which always starts at a specific node and performs a recursive depth first search by using the rules mentioned in IV-B to find child nodes. We call this algorithm *runThroughSubtree*. For its assignment in propagation, it must get two different kind of information:

1) The propagation function that should be performed.
2) An identification of the right(s) it has to deal with.

The propagation functionalities are implemented within five different classes, according to the five kinds of propagation. Each of these classes extends from the abstract superclass 'PropagationStrategy'. This superclass instructs the use of two different methods, which must be overwritten:

- *allowTermination*: Decides on which nodes the recursion can be stopped.
- *handleObject*: Decides which actions have to be performed on the visited nodes.

Both methods use a so called context, which is implemented as a list of rights that can be assigned to the runThroughSubtree algorithm. During the recursion, this context may be changed by handleObject to fullfill the rules mentioned in IV-B. This is done by a compare function, which solves all overlappings of context rights with local explicit rights. In a similar manner, the local explicit and/ or inherited rights can be adjusted to the context list.

D. Inherit rights

This algorithm performs the following steps:

1) Take a look if the inheritance flag on the target WasabiObject o is already activated. If so, the algorithm stops.
2) Load all rights of the superordinated WasabiObject o' in the context. Now this context and an object of the class *Inherit* is assigned to the propagation function.
3) If the propagation function terminates successfully, the inheritance flag is set.

In the class *Inherit* both functions which are instructed by the PropagationStrategy class, are defined:

- *allowTermination*: If the context is empty or the inheritance flag is not set (except in the root) the algorithm terminates.
- *handleObject*: Adjust the context to the local explicit rights by the rules mentioned in IV-B. Then propagate the context as inherited rights.

E. Disinherit rights

This algorithm performs the following steps:

1) Take a look if the inheritance flag on the target WasabiObject o is already deactivated. If so, the algorithm stops.
2) Load all inherited rights of o in the context. Now this context and an object of the class *Disinherit* is assigned to the propagation function.
3) If the propagation function terminates successfully, the inheritance flag is deactivated.

In the class *Disinherit* both functions which are instructed by the PropagationStrategy class, are defined:

- *allowTermination*: Similar to V-D.
- *handleObject*: Adjust the context to the local explicit rights by the rules mentioned in IV-B. Then remove all local inherited rights which compare with the context.

F. Remove rights

This algorithm performs the following steps:

1) Load the should be deleted right(s) into the context.
2) Compare the context with the explicit rights of the target object. Every context right with no equivalent should be removed.
3) Assign the (perhaps shortened) context to the propagation function and also an object of the class *Remove*.
4) Look into the superordinated object of the target and load all rights, which perform a collision with the (perhaps shortened) old context into a new context.
5) Assign the new context to the propagation function and also an object of the class *Inherit*. The root of this function is the target object.

In the class *Remove* both functions which are instructed by the PropagationStrategy class, are defined:

- *allowTermination*: Similar to V-D.
- *handleObject*: Adjust the context to the local explicit rights by the rules mentioned in IV-B. Then remove all local explicit (in the root) or inherited rights which compare with the context.

G. Grant rights

This algorithm performs the following steps:

1) Load the right(s) into the context.
2) Assign the context to the propagation function and also an object of the class *Grant*.

In the class *Grant* both functions which are instructed by the PropagationStrategy class, are defined:

- *allowTermination*: Similar to V-D.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

- *handleObject*: Adjust the context to the local explicit rights by the rules mentioned in IV-B (not in the root). Adjust the local explicit rights to the context (only in the root). Adjust the local inherited rights to the context. At least, propagate the context as explicit rights (in the root) or inherited rights.

H. Reset rights

This algorithm performs the following steps:

1) Assign an object of the class *Reset* to the propagation function.

In the class *Reset* both functions which are instructed by the PropagationStrategy class, are defined:

- *allowTermination*: The algorithm terminates if the initiator has got no GRANT on the local node.
- *handleObject*: If the inheritance is not activated, set the flag. Remove all local, explicit rights by using the Remove-function (see V-F).

VI. Authorization Functions

The last chapter has dealt with the right management and its consistency. Now we show how the declared rights can be used for authorization functions.

Altogether, there are three different algorithms for the check if a user owns a specific right on an object. At first, the certificate check is performed. If no relevant certificate with a required entry can be found, WasabiBeans uses the fast check. But this fast check may only be performed if there are no time rights on the target object. Otherwise, it has to use the slow check.

A. The Certificate Check

A certificate is always generated or updated, if a specific user right was checked. For this reason, it is not important if this check has lead to an allowance or a forbiddance. This means, a certificate should not always grant an allowance but saving the result of the last check. The collectivity of all certificates forms a cache which should fasten the authorization process. Its advantages on speed result from the following points:

1. Only the rights which are used more often are regarded.
2. The expensive checks for group membership needn't be performed.

Point 2 leads to the conclusion that a certificate is only declared on a specific user and never on a group. This is because the user is the only subject of the authorization process and not its groups. For this reason, the certificates are saved within the WasabiUser objects. We use hashmaps which form relationships of the kind WasabiObject - WasabiCertificate. This leads to another advantage:

3. There is no need to run through a big list of right objects: Instead of $O(n)$, the access on a hashmap often has got an effort of $O(1)$.

A certificate is always removed if the corresponding rights are changed. Another reason is the change of the users group memberships. If such a change is performed, all certificates of the user must be removed because an exact check for relevant certificates would be too expensive.

An important point of our relationship WasabiUser - WasabiObject - WasabiCertificate is the fact that the kind of permission is not involved. For this reason, a WasabiCertificate is implemented as the sum of all seven possible permissions of a WasabiUser - WasabiObject relationship. These seven permissions are combined in one single value. This value has got an initial 0. If a permission was checked by fast or slow check, it gets the following extension:

- Allowance: Add $3^{permissionNumber-1}$
- Forbiddance: Add $2 * 3^{permissionNumber-1}$

If a certificate check is performed, the algorithm has to look at the specific right tertiary position. On a value of 1 the algorithm grants an allowance, a value of 2 leads to a forbiddance. However, a 0 has got the meaning that this permission wasn't checked already and so one of the other algorithms must be used. Before this, the algorithm also considers the timestamp of the certain certificate. If it is smaller than or equal to the actual system time, the certificate must be dropped. A certificate may always get such a timestamp, if the Slow Check was used (see VI-C).

B. Fast Check

This algorithm is very simple: It runs through the list of WasabiACLEntrys and stops if it finds the first relevant allowance or forbiddance. As a condition, the rights must be sorted by their hierarchy level:

- Level 4: User forbiddance
- Level 3: User allowance
- Level 2: Group forbiddance
- Level 1: Group allowance

Such a sorted form arrangement can be granted by the propagation functions. If no relevant right was found, it is taken as a general forbiddance. If the Slow Check was partially performed before, the algorithm must eventually consider the value $maxTime$ for the generation or update of the certificate (see VI-C).

C. Slow Check

This complete check is always needed if at least one time based right exists in the target WasabiObject. The reason for this consists of the fact, that the hierarchy mentioned in VI-B can't be used: If a right of high level is already found, it may be covered later by a colliding time right of a low level which lies deeper in the list.

The algorithm uses the following steps:

- **Prescan:** Find all relevant time rights[6] and insert them into a special list. Generate the value $maxTime$, which is the lowest start time of all time rights which fit for the subject and needed permission and which $startTime$ is higher than the actual time.

[6] A time right is relevant if it fits for the subject, the needed permission and the actual system time

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

- **Postscan:** If no relevant time right was found, the fast check may be performed. Otherwise, find all relevant, unlimited rights and insert them into a second list. But such an insert is only allowed if the right has got no overlapping with the list of relevant time rights.
- **Final Scan:** Run through both lists. Every right is saved within a special variable, if its hierarchy level is higher than the hierarchy level of the right saved before. Having reached the end, the state of the last saved right is taken (allowance or forbiddance). If both lists are empty, it is taken as a general forbiddance.

The certificate handling has got the extension that the value $maxTime$ must also be considered. If a new certificate is generated, it gets $maxTime - 1$ as a timestamp. Otherwise, if the algorithm has to update an existing certificate, its timestamp is set to $maxTime - 1$, if $maxTime - 1$ is smaller than the existing timestamp.

The figure 5 shows an example for a slow check:

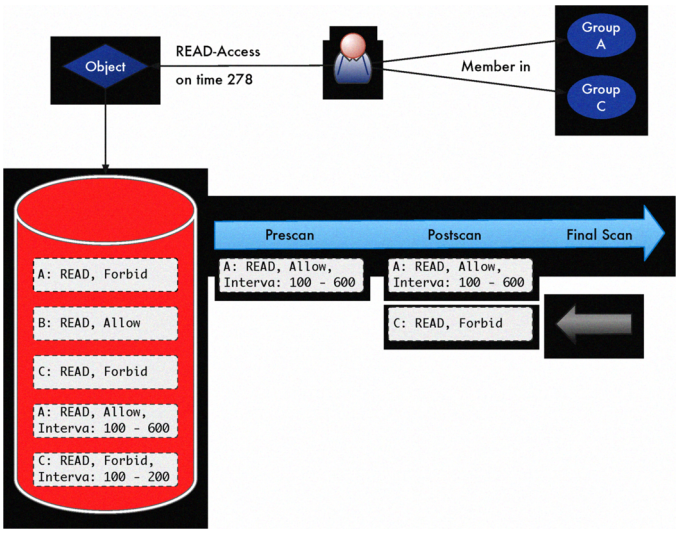

Fig. 5. An example for a slow check

VII. Scenarios

A. Scenario 1 - The change of rights

This example of use deals with the change of explicit rights. For this case, some rights of a user on a special container shall be added and others shall be removed. Along with this change comes an affection of the subordinated objects. On a system which works without inheritance, this could cause the effect that some substructures of these objects may be overwritten if they collide with rights which are changed above. This means that the existing right structure of this user becomes obsolete. For example, there is a forbiddance on a subcontainer which now becomes overwritten by an allowance from above. But such an overwriting may be undesirable if the assertice forbiddance should continue existing. For this case, it must be renewed after the operation or it must from the first be determined where the recursive right allocation should stop. Especially on folder structures with a very complex right structure, where many similar cases are given, the administrative effort may increase strongly.

Up to this, the inheritance of rights has got the advantage, that it is possible to change the rights of a user on a space or a container without destroying the subjacent structure. But there is also the option, that such an absolute adaptatio may be done (by the reset function) if it is desired. From this it follows that the administration becomes more flexible by the concept of inheriting rights than by a more simple scheme.

Eventually, after such a right granting it can be useful to know if there also exist rights in the below subtree which affect the specific user. As mentioned before, these rights would cover the new right which was granted above. In this case, the administration environment may intervent and show the administrator that these substructures and in which locations they exist. Because of this information it is possible to make an easier decision, if and which substructures have to be removed.

B. Scenario 2 - Research and presentation

There is a number of users which shall get some documents for research. These documents were sorted by their theme into several containers. Now the system administrator wants to create a new group, which shall get the required access rights. Possibly this could be created as a subgroup which superordinated group already owns a great amount of these access rights. Then it would only be necessary to declare the missing rights to the new group (at least there must be the VIEW and READ). At least, all users which come in question for a research are added to this group.

In addition to this, the members of the research group may get access on another container structure, where they can document their conclusions. But instead of declaring these rights to every single user, the administrator only needs to enhance the right amount of the group. In addition to a simple read access (VIEW and READ) an upload (INSERT) is also granted. It could be considered that a general write access (WRITE) should also be permitted. For this reason, there is the advantage that it would also be possible that an user may advance the uploaded conclusion of another (only the creator, in this case the uploader of a document gets all access rights on it as initial). A disadvantage of such a decision is, that it it also possible to delete hardly created results.

Having released this structure, we have got the fact that a single user has abused its access rights. For this reason, they should be denied from him. Because of the fact that a direct access denial is not possible, the administrator uses s specific user forbiddance, which is mightier than all allowances the user got from his group relationship(s). Under circumstance, this denial could be limited on time, whereby it is like a time based penalty.

C. Scenario 3 - Publishing

In this scenario a container structure should be moved from a WasabiRoom A into another WasabiRoom B. As the right structure from space A is very restrictive, space

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377

B grants more liberties. This means, the data of space A lies in a relative secure area. By moving a part of it into space B, it can be published to a greater publicity. Here the concept of inheritance comes into use: Every right on space B is automatically assigned to the moved container structure, without the need of an explicit recognition. In this manner, existing structures may be reused.

Now we have got the case that the container structure itself contains explicit rights, which are also moved. This could be a problem if there are forbiddances which undermine the structure of space B which depends on openness. At this point, the administrator must intervent: He has got the option to break all substructures of the moved container by using the reset-function. But thereby we still have got the possibility that he doesn't know the existence of these substructures. For this reason, the administrative surface may warn him that they subsist, so that it is easier for him to make such a decision.

VIII. Conclusion

In this paper we presented a flexible authorization infrastructure for cooperative work environments. Furthermore we introduced WasabiBeans, a framework for implementing applications in the scope of computer supported cooperative work. We exemplified, that an authorization infrastructure must fulfill special requirements to be applied to cooperative systems, that are characterized by dynamic and changing work processes. Especially the concept of assigning privileges by the use of inheritance assists both, setting up new environments for collaborations as well as common work processes like exchanging documents among different users. Another reason for such a sophisticated approach is justified by the advantages that come along with the new propagation functions. These functions guarantee that the rights of moved or copied objects are always adjusted to the new location correctly.

We came to the conclusion, that a set of permissions is necessary, that support cooperative work. For instance the COMMENT permission can be used to permit or deny users having discussions on an object. The VIEW permission involves the possibility to make special objects invisible. Thereby users may not notice them and the protection of privacy can be granted. Although the infrastructure is flexible and suitable for various fields of application, users of the final application benefit from intuitive handling. In difference to other systems the access rights in WasabiBeans are specified directly on the objects they are belonging to. This is beneficial when planing systems with migration functionalities, since all rights are specified on the object itself and cannot be lost.

Last but not least the focus of this paper was on performance issues of the authorization infrastructure. Hence, we use the propagation algorithms to speed up security checking functions. In addition sorted lists and certificates ensure very efficient inspection of already investigated objects and spare the more complex checking functions.

References

[1] R. Capurro, *Ethik im Netz*. Franz Steiner, 2003, no. 978-3-515-08173-3.

[2] ——, *Zwischen Vertrauen und Angst. Über Stimmungen der Informationsgesellschaft*. Springer Berlin Heidelberg, 2008, no. 978-3-540-77669-7.

[3] K. Stark, J. Schulte, T. Hampel, E. Schikuta, K. Zatloukal, and J. Eder, "Gatib-cscw, medical research supported by a service-oriented collaborative system," in *CAiSE 2008 - 20th International Conference on Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, Z. Bellahsene and M. Léonard, Eds., vol. 5074, no. 978-3-540-69533-2. Montpellier, France: Springer, June 2008, pp. 148–162.

[4] B. Eßmann, T. Hampel, F. Goetz, and A. Elsner, "Embedding collaborative visualizations into virtual knowledge spaces," in *7th International Conference on the Design of Cooperative Systems*, France, Provence, 5 2006, p. 3340.

[5] T. Hampel, H. Selke, and R. Keil-Slawik, "Semantische räume - von der navigation zur kooperativen wissensstrukturierung," in *Mensch und Computer 2004: Allgegenwärtige Interaktion*. Oldenbourg Verlag, 2004, pp. 221–230.

[6] T. Hampel and R. Keil-Slawik, "sTeam - designing an integrative infrastructure for web-based computer-supported cooperative learning," in *Proceedings of the 10th International Conference on World Wide Web*, 2001, pp. 76–85.

[7] T. Licht, L. Schmidt, and H. Luczak, "Goal awareness in distributed cooperative work settings," in *Human Factors in Organizational Design and Management VII (Aachen 2003)*, H. Luczak and K. J. Zink, Eds. Santa Monica: IEA Press, 2003, pp. 329–334.

[8] W. Prinz and T. Gross, "Ubiquitous awareness of cooperative activities in a theatre of work," in *Fachtagung Arbeitsplatzcomputer: Pervasive Ubiquitous Computing*, A. Bode and W. Karl, Eds. APC, October 2001, pp. 135–144.

[9] J. Schulte, T. Hampel, T. Bopp, and R. Hinn, "Wasabi framework – an open service infrastructure for collaborative work," in *SKG '07: Proceedings of the Third International Conference on Semantics, Knowledge and Grid*, no. 0-7695-3007-9. Xi'an, China: IEEE Computer Society, October 2007, pp. 242–247.

[10] ——, "Wasabi beans – soa for collaborative learning and working systems," in *DEST '08: Proceedings of the Second IEEE International Conference on Digital Ecosystem and Technologies*. Phitsanulok, Thailand: IEEE Computer Society, February 2008, pp. 177–183.

[11] T. Hampel and P. Heckmann, "Deliberative handling of knowledge diversity the pyramid discussion and position-comentary-response methods as specific views of collaborative virtual knowledge spaces," in *Proceedings of Society for Information Technology and Teacher Education, 16th International Conference Annual, SITE 2005*, Arizona, USA, 2005.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8377
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8377