Experimenting with a Flexible Awareness Management Abstraction for Virtual Collaboration Spaces

Ricardo Jota, Jorge Martins, António Rito-Silva and João Pereira
INESC-ID/Technical University of Lisbon Rua Alves Redol nº 9, 1000-029 Lisboa, PORTUGAL
{Jota.Costa, Jorge.B.Martins, Rito.Silva, Joao}@inesc-id.pt

ABSTRACT

The awareness management problem is still very far from solved. It is difficult to create an abstraction that's flexible enough to be used in the wide range of applications that deal with Awareness management problems. This paper describes a generic object-oriented abstraction for the problem of awareness management in Collaborative Virtual Environments (CVEs). The described abstraction allows us to use different types of awareness information and awareness management policies. It is shown how the defined abstraction was applied to the conference table problem, what qualities where observed and how different variations of the same problem are solved using an incremental solution.

1. INTRODUCTION

Awareness is a very important concept in CSCW systems. As stated in [6] "awareness is an understanding of the activities of others, which provides a context for your own activity". Collaborative Virtual Environments (CVEs) are networked virtual environments used to support collaborative work. Users are represented graphically within the environment and can perceive other user actions through their graphical representation. In these systems, awareness information is all the information about existing objects and users within the system. This information includes user and object graphical representation, the sounds produced by users, and user actions. These systems also have the characteristic of aiming to support a large number of simultaneous users (in the order of the tens of thousands). In the presence of a large number of users the amount of information that must be processed by each user can be overwhelming. As such, it is usually necessary to manage the amount of information that must be processed by each user. This is called awareness management. The goal of awareness management is to allow each user to only process the information that is relevant for him. Some of the existing systems use awareness management as a mechanism for reducing network bandwidth and increasing their scalability, while others use awareness management to promote user collaboration by using it to scope user interaction. The work described in this paper was done in the context of the MOOSCo (Multi-user Object-Oriented virtual environments with Separation of Concerns) project. MOOSCo proposes a software engineering separation of concerns approach for the development of Multi-user Virtual Environments. For each of the different aspects of these systems, different concerns are identified. The system functionality is obtained by composition of the solutions defined for each of the concerns. Further details about the MOOSCo approach can be found in [2][3]. Section two describes the related work. Section three presents the awareness management abstraction,[1]. The Experimentation section describes several solutions using the MOOSCo framework, and their use of the abstraction to support different awareness management policies in the conference table problem. Section five evaluates our approach to the awareness management problem. Finally, section six presents our conclusions and future work.

2. RELATED WORK

Awareness management is a very important issue in CVEs. It is used as a mechanism of regulating the amount of information each user must process. Awareness management helps collaboration between users, by suppressing all the awareness information about users and objects that are not relevant for the current users' collaborative task. Awareness management also has an important role in the scalability of this kind of systems. By limiting the amount of information that must be processed by each user, awareness management can be a very effective mechanism for reducing resource usage, like network bandwidth and computer processing power.

Given the importance of awareness management, different policies have been used depending on the systems requirements and goals. In RING [7] users are only aware of the objects they can see. This policy is adequate for environments that contain several visual barriers such as walls and doors, but it performs badly in densely populated open space environments. SPLINE [4] partitions the environment in spatial regions called locales. Each user is aware of all the objects in the current locale and in the immediate neighbors. The partitioning of the environment is a very flexible mechanism for structuring the virtual environment. In SPLINE each locale defines its own independent coordinate system. The virtual environment results from the connection of several locales. Each connection between two locales defines a 3D transformation that describes the relations between the locale's coordinate systems, which allows the creation of non-Euclidean environments. This approach also eases the extension of the environments, since it is only a matter of defining new locales and connecting them to the existing ones. Finally, the locales approach also provides an effective mechanism for controlling awareness. Allowing the awareness of the adjacent locales gives users the notion of spatial continuity, increasing at the same time the system scalability. Unfortunately, SPLINE only provides this built-in policy. However there are situations where different policies could be more useful. For instance, one could be interested only in the current locale, or interested in the current locale and a small subset of the adjacent locales. NPSNET [10] divides the environment in fixed-size regions called cells and each user defines an area of interest through an aura. Users are only aware of the cells their auras intersects. The size and shape of the cells were chosen taking into account the target application domain, military simulation. MASSIVE-1 supports the spatial model of interaction [5]. In this model users define auras and interaction between two users can only happen when the two users' auras intersect. The model also uses the concepts of focus and nimbus to compute the awareness level that a user can have of another user or object. The focus represents an user interest in a particular medium. The nimbus represents an observed object's projection in a particular medium. The awareness level that an object A has of an object B in a particular medium M is a function of A's focus and B's nimbus in M. In MASSIVE-2 [9] the spatial model of interaction was extended with the third-party object concept which represents objects that can affect other object's and user's awareness levels by changing their values of aura, focus and nimbus. The spatial model of interaction is perhaps one of the most complex and flexible awareness management models. It is suitable for controlling user interaction in large-scale virtual environments. However, the model is too focused on the spatial aspect making it difficult to manage awareness using semantic or organization considerations. In MASSIVE-3 [8] an extension to the SPLINE model was adopted. In this extension the environment is also divided into locales and the locales can be connected through boundaries. Each locale can have several aspects, each representing a certain type of awareness information. The awareness management is performed, by selecting the locales and corresponding aspects that are relevant to a particular user. The system allows the programmer to select, or even adapt, the policy. Since locale aspects can be arbitrarily defined, it is possible to support awareness management taking into account organizational associations between the objects of an environment. Due to the existence of different application requirements it is necessary to support different awareness management policies. All the mentioned systems only offer support for a particular policy or family of policies. MASSIVE-3 allows some degree of adaptation, by letting the programmers choose the policy that selects the relevant locales to a particular user. However, the adaptation is confined to a particular kind of awareness policy, based on locales and aspects. It is not possible to use different policies. This problem is normally due to the lack of proper design abstractions that are, not only able to solve the problem at hand, in this case awareness management, but also able to support the several variations that exist for the problem's solution. To deal with this problem, this paper presents an object-oriented awareness management abstraction that is flexible enough to support different awareness management policies. Instead of trying to provide a one-size-fits-all solution for awareness management, the proposed abstraction allows different policies to be defined, and programmers have to choose the most appropriate policy for the application being developed.

3. AWARENESS MANAGEMENT ABSTRACTION

This section describes an object-oriented abstraction for awareness management in CVEs. It is not the goal of the proposed abstraction to define a generic model for awareness management that can be used for all kinds of CVEs systems. Instead the abstraction aims to provide a common framework upon which different solutions for awareness management can be built. To be able to achieve this goal it is necessary that the abstraction is flexible enough to capture the variations that exist in the different solutions for the problem. The description of the Awareness Management abstraction is divided in three sections. First we present a general description of the awareness management problem. Second we describe different variations that a flexible solution for awareness management should support. third we present our

solution, i.e. the proposed object-oriented abstraction for awareness management. The solution presentation consists of: a description of the structure and the elements that are part of the abstraction; a description of how these elements collaborate to solve the problem at hand; and a description of how the abstraction supports the variations that were identified.

3.1 Problem

One of the issues of awareness management is what type of information must be considered for awareness management purposes? For instance, the sound produced by a certain user A can be used as awareness information. In this case user *B* will be aware of *A* when he can hear him. The geometric appearance of objects and users can also be considered as awareness information. Each user becomes aware of objects and users from the moment he/she sees them. Even the user's actions can be used as awareness information. For example, user *A* may be able to see user *B* but be unable to understand the actions he/she is executing due to the distance between them. After approaching user B, user A may then be aware of the actions user B is performing. Once the information used for awareness management is defined, it is necessary to define how the users declare their interest in certain types of information. Although there might exist different types of awareness information, a user may, at a certain moment, only be interested in a particular type. Once the awareness information and mechanisms by which the users express their interest are defined, it is necessary to guarantee that each user only receives information that is relevant for him. There are several policies to manage who should receive a certain type of information. For instance, the proximity to the information source may be a way to define who may receive it. Moreover, the existence of visual barriers may be used to determine who should not receive certain visual information.

3.2 Variations

A solution for awareness management must support the following variations:

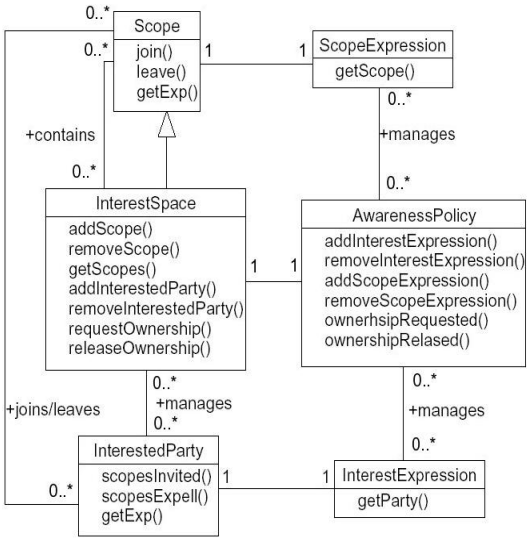
Awareness Information Definition. The awareness information sould be defined in accordance with the application requirements. The choice of awareness information can be determined by the applications functional requirements, e.g., collaboration, or by non-functional requirements, e.g., the system has to support a large number of simultaneous users.

Different Awareness Management Policies. Different awareness management policies represent different ways of computing the awareness information that should be received by each user. Each application should be able to choose the most appropriate awareness management policies to apply in their contexts. Moreover, they should be able to define their own awareness management policies that take into account the application specific requirements.

3.3 Solution

The main characteristic of the proposed solution for the awareness management concern is the identification and separations of all entities that are present in the awareness management problem. These entities are: The sources of awareness information; The consumers that are interested in receiving this kind of information; And the policy that is responsible for disseminating the information to the interested consumers. This way the management policy can be changed, independently from the awareness information and the behavior associated with handling that information. Moreover this separation also allows the use of different types of awareness information, independently of the awareness management policy used.

3.3.1 Structure and Participants



The abstraction Awareness Management has the following participants:

Scope. Represents a source of awareness information..

ScopeExpression. Represents expressions that describe a certain information. Each Scope has a scope expression associated that describes information it represents.

InterestedParty. Represents an entity interested in a certain type of awareness information. It must be associated with a scope to have access to that information. The association and dissociation of interested parties to scopes is managed by the awareness management policy.

InterestExpression. Represents the interest of a party in a certain type of awareness information.

InterestSpace. Represents a space that contains several scopes for the same type of awareness information. Interested parties are registered in interest spaces by indicating their interest expressions. An interested space can also play the role of a scope. This way, it is possible to define hierarchical interested spaces that use different awareness policies at each hierarchical level. The awareness management of the whole structure results from the awareness management of its constituent parts.

AwarenessPolicy. Represents an awareness management policy. A policy is responsible for determining the interest expressions matched by each scope expression. Every time a match is detected, the policy informs the corresponding interested party that it should be associated with the scope. When the match ceases the policy informs the interested party to dissociate itself from the scope. Usually, the scope and interest expressions are dependent of the awareness management policy. However, the interested parties and the information scopes are independent of the expressions and policy being used. The choice of which interest expressions and scope expressions to use depends of the interest space awareness policy.

4. EXPERIMENTATION

The *Awareness Management* abstraction described in this paper was implemented in Java™ as an object oriented micro-framework. This micro-framework is part of a larger framework called MOOSCo that supports the development of CVEs. The MOOSCo framework was developed using a separation of concerns approach. An abstraction was defined and implemented for each concern. The support for CVES was obtained by composition of the concern implementations. These concerns are described with more detail in [1]. In this paper our goal is to show that the *Awareness Management* abstraction has the following qualities:

Expressiveness power. The awareness management abstraction must be able to produce a solution to a large set of problems. The abstraction must have enough flexibility so that it can be applied efficiently in different problems. Moreover, its flexibility must allow programmers to solve each variation of the same problem with minimal changes from the generic solution for that particular problem. The framework must be flexible enough to allow the concerns to interact and therefore allow solutions that include cooperation of concerns to be implemented.

Incremental development. One of our goals is to promote incremental development by allowing programmers to change the instantiated composition, starting from the simpler ones and moving incrementally to the more complex ones. The ideia is to start with a basic solution to the problem to solve and then build upon that solution to achieve more complex solutions to variations of the problem.

Easy to use. To develop software using a framework, the framework should not overhead programmers with hard to use interfaces. It must be intuitive and easy to use with the rest of the code the programmer must develop.

These are the goals that guided our experimentation and the qualities that are present in the abstraction and the framework.

In the remainder of this section we present an experiment that show how our solution achieves these qualities. We start with a simple example and we'll move incrementally to more complex ones.

4.1 Conference table – basic implementation

The environment of our experiment consists on a conference table, around which users can sit. In this experiment awareness management controls the interaction between the persons that sat at the table and the remaining persons of the environment. Persons at the table cannot interact (chat) with users outside the table. However, depending on the awareness configuration, users outside the table may or may not "hear" what the users at the table are saying.

For our first example we will use the following rules to describe the conference table environment:

1. There are two kinds of persons, the ones sitting at the table, called invited speakers, and the ones away from the table, called audience. The audience members can have a microphone that they use to ask questions to the invited speakers.
2. Everyone is able to ear the invited speakers.
3. Everyone is able to ear audience members with microphone.
4. Each audience member is able to ear the other audience members according to their proximity.
5. The invited speakers are not able to ear the audience members without microphone (note rule number 3.).

4.1.1 Solution

This problem has two kind of interactions:

- The one between invited speakers (and audience members with microphone) and all the other people
- The one between audience members

This indicates that the problem can be effectively solved with two *InterestSpace*. The first manages the interaction between invited speakers and all the other people, and implements a policy "all-see-all". The second one manages the interaction between the audience members. The second *InterestSpace* implements an Aura policy in which each person sees others according to their proximity.

Another remark is that an audience member with a microphone behaves exactly like an invited speaker. This

can be implemented by giving the audience members the property of having a microphone. When this property is valid then the audience member behavior is just like an invited speaker and just requires inserting the audience member's *Scope* in the invited speakers' *InterestSpace*.

4.1.2 Implementation details

Each person defines an *InterestedParty* which represents her interest in what others are saying. An *Interested Party* includes an *InterestExpression.* Recall that the *InterestExpression* defines the interest of an *InterestedParty* in a certain type of awareness information. Each person can be a source of information therefore, the people have to define a *Scope* which acts as the source of the speech information. The *Scope* includes a *ScopeExpression*. That describes the sound reach of its associated person.

This way we can define exactly what each person hears and to whom each person talks to, e.g, everybody or just the audience members that are near the concerned person.

The environment is divided in two *interestSpaces*:

Invited speaker InterestSpace – This *InterestSpace* enables everyone access to the information produced by the invited speakers. This *InterestSpace* will use an "all-see-all" policy. In this policy every party knows every scope present in this *InterestSpace*. Therefore, the following parties and scopes are inserted in this InterestSpace:

- The parties of invited speaker and audience members parties;
- The scopes of invited speaker. This way everyone ears them;
- The scopes of audience members with a microphone.
- **Audience member InterestSpace** – This *InterestSpace* manages the interaction between audience members. As we said previously, audience members can interact if they are physically close. To implement this requirement, this *InterestSpace* will use an Aura Policy. This policy defines *InterestExpressions* and *ScopeExpressions* as auras centered at each user position. When an *InterestExpression* of a user A intersects a *ScopeExpression* of a user B, user A party joins user B scope. All the audience member parties and scopes are placed in this *InterestSpace*.

4.1.3 Conclusions

We can apply our awareness management abstraction in several ways to solve this problem. [1] describes another possible way to solve the conference table problem.

The *Aura* policy used in this solution is similar to the spatial model of interaction used in MASSIVE-1 and can be easily modified to be compatible with NPSNET [10]. To be compatible with NPSNET [10] one would use square auras in a discrete referential, this would be equal to the fixed regions called cells. As in NPSNET users define areas of interest trough auras it would be compatible with our *Aura* policy. Our "all-see-all" policy could be modified to be compatible with RING [7] just by adding rules that express that if a user sees an object but had another object intersecting the direct line between the user and the first object then the user would not be able to see the first object. By having such an expressiveness power our framework covers most of the related work with just two policies with minor adaptations. The microphone, in this example is treated like a property. That extends the information range. For instance, if an audience member has a microphone he can be heard by the invited speakers. This kind of property that modifies an user awareness is used in policies based on properties. In this example, the *Aura* policy used is composed with this microphone property to allow the users to extend their awareness range when desired.

In the rest of this section, we present several iterations of the conference table problem by adding additional requirements and apply an incremental development where each solution is based on the previous one. This shows the quality of incremental development of our awareness management abstraction. Each iteration exercises a different part of the abstraction, globally demonstrating the expressiveness power of the abstraction.

4.2 Conference table – iterations

Having solved the conference table with a solution that uses the MOOSCo framework, now we will introduce several requirements that aren't addressed by the previous solution. Each iteration will be achieved by incremental development of the previous solution.

4.2.1 First iteration - Invited speakers with just one microphone

In this iteration, we introduce the following restriction:

- Only one invited speaker may be heard by the audience members at a giving moment

At first sight, the basic solution would solve this, but notice that by allowing just one invited speaker to be heard by the audience members, we still allow other invited speakers to speak among themselves.

4.2.1.1 Solution

Unlike the last solution, we now have three interactions to deal with:

- Between invited speakers
- Between the invited speaker allowed to speak and the audience members
- Between audience members

To solve this iteration, we divide the *Invited speaker InterestSpace* used in the last solution into two *InterestSpaces.* Once again, the number of interactions we have to deal with equals the number of *InterestSpaces* the solution has.

4.2.1.2 Implementation details

Each person is defined as described in the last solution.

The environment is divided into three *InterestSpaces*:

Invited speaker InterestSpace – This *InterestSpace* enables invited speakers to have access to the information provided by other invited speakers. This *InterestSpace* uses an "all-see-all" policy. The scopes and parties of the inivited speakers are placed in this *InterestSpace*.

Microphone InterestSpace – This *InterestSpace* allows everyone to have access to the information expressed by the invited speakers or audience members with the microphone. This interestSpace inclues the all the parties defined and the scope of the person that has has the microphone. It applies the "all-see-all" policy.

- **Audience member InterestSpace** – This *InterestSpace* is defined as before.

4.2.1.3 Brief discussion

The most visible change form the previous iteration is the number of *InterestSpaces* required to solve the conference table problem. We could present a solution that would not change the number of *InterestSpaces*. For instance, we could, instead of inserting the audience member parties in the invited speakers *InterestSpace* as in the basic implementation do the opposite, insert the invited speakers scopes (expresses the information) into the audience members *InterestSpace*. This would of course bring another problem, when an audience member possesses a microphone one would have to insert his scope in the invited speakers *InterestSpace* and change his scope expression in the audience members *InterestSpace*. This isn't a trivial matter, as there aren't any restrictions to the "geometry" of an *InterestSpace*. So by choosing to increase the number of *InterestSpaces* we demonstrated some expressiveness power of the framework and the small number of changes we have to do to the previous implementation while keeping the solution easy to understand.

4.2.2 Second iteration - Invited speakers interact by proximity

In this iteraction we change the way the invited speakers interact among themselves. Instead of, allowing invited speakers to hear every other invited speaker we apply the following rule:

- An invited speaker hears other invited speaker when
  - The invited speaker is in his proximity
  - The invited speaker has a microphone

4.2.2.1 Solution

In the previous solution we had three *InterestSpaces.* One of those *InterestSpaces* manages the interaction between invited speakers. To solve this new rule all we have to do is change that *InterestSpace* policy to an *Aura* policy.

4.2.2.2 Implementation details

Each person is defined as described in the last solution.

The environment is divided into three *InterestSpaces*, as in the last solution. In this solution, we only change the definition of the Invited speaker InterestSpace**.** Now this *InterestSpace* enables invited speakers access to the information expressed by other invited speakers if they are near. This *InterestSpace* uses an *Aura* policy. The scopes and parties of all invited speakers are inserted in this *InterestSpace*.

4.2.2.3 Brief discussion

This iteration clearly shows the flexibility of the framework. At code level all we have to do is change the code line where we create the invited speaker *InterestSpace* policy to associate it with an *Aura* policy.

4.2.3 Third iteration - Invited speakers can comment between them

In the previous iterations an invited speaker could not say something to someone without, at least, everyone within range hear it too. In this iteration, an invited speaker is able to specify who is supposed to hear what he is saying. Note that having or not a microphone does not matter. If an invited speaker wants to comment something, just the ones he chooses would hear them. Comments can only be done between invited speakers.

4.2.3.1 Solution

This iteration brings two additional problems. The first one is how an invited speaker with a microphone comments something. The second is how one expresses who's supposed to hear and who's not. To solve the first problem we have to change the invited speakers' *InterestSpace*. The scope used in this *InterestSpace* must be different from the one used in the microphone *InterestSpace,* this means that each invited speaker is composed of two *scopes* and a *InterestedParty*. The *InterestSpace* policy, and scopes, would have to take into account the target invited speakers.

4.2.3.2 Implementation details

Each audience member is defined as described in the last solution.

Each invited speaker is defined an *InterestedParty* and two *scopes*. One *scope*, called comment scope, is inserted in the invited speaker *InterestSpace* and expresses comment information. The second one, called normal scope, is inserted into the microphone *InterestSpace* when the invited speaker has the microphone.

- The environment is divided into three *InterestSpaces*. In this iteration we have to change the definition of the invited speaker *InterestSpace*. The other two *InterestSpace*'s are identical to the previous solution. The invited speaker *InterestSpace* enables invited speakers to have access to the information expressed by other invited speakers. Now this *InterestSpace* uses an *Aura* policy that expresses target invited speakers. When an *InterestExpression* of a user A intersects a *ScopeExpression* of a user B and the *ScopeExpression* expresses that the user A is a target user for this

information, user A's party joins user B's scope. The parties and comment scopes of all invited speakers are added to this *InterestSpace*.

4.2.3.3 Brief discussion

Having the invited speakers composed by two *scopes* and a *party* is not an odd or a rare option. To solve this kind of problem where the same entity broadcasts two distinct types of information it is necessary to have different scopes, with different expressions that describe that information scopes. Our abstraction is sufficiently expressive to support this capability.

In this solution we added classes that did not exist in the previous solution (the new scope) and changed, once more, the policy of one *InterestSpace* to cope with the new requirement. We where able to use most of the code developed to solve the previous solution. We only had to modify three code lines of the previous implementation and add the new classes to implement the new solution. This shows the incremental development quality of our abstraction.

4.2.4 Fourth iteration - Invited speaker microphone has a range

In the previous iterations we treated the microphone as a property of the invited speakers or the audience. To improve the example we will treat the microphone like a real object and not a property. This can be described in the following rule: The invited speakers can use (talk to) the microphone only if they are in the microphone proximity. Otherwise the invited speakers can only comment among themselves.

4.2.4.1 Solution

With this example we take the following conclusions about the qualities referred above:

To treat the microphone like a real object we need to define it as an *InterestedParty* (his range) and a *Scope* (that represents the sound system range connected to the microphone). The interaction between invited speakers and audience members is done through the microphone.

Unlike the previous solution the invited speaker does not need to define two *scopes*. Now, the second *scope* that was inserted in the microphone *InterestSpace* is replaced by the microphone *scope.*

In this solution we keep the architecture of the previous solution, maintain the same number of *InterestSpaces,* but change where each *Scope* and *InterestedParty* is inserted. In this solution the only scopes inserted in the microphone *InterestSpace* are the microphones *Scopes*. The other *Scopes* interact with the microphone *InterestedParty* and not directly with the listeners *InterestedParty*

4.2.4.2 Implementation details

The microphone is defined using a *Scope* and an *InterestedParty*. The *Scope* represents the information the microphone wants to express and the *InterestedParty* represent the interest the microphone expresses to hear all sounds that are in its range.

Both audience members and invited speakers are composed by a *Scope* and an *InterestedParty* likewise to what we have defined in the previous iteration. The *Scope* represents what that person says and the *InterestedParty* represents what the person can hear.

The envioremnt is, again, represented by three *InterestSpaces.* The audience member *InterestSpace* is equal to the one defined in the previous iteration. The invited speaker and microphone *InterestSpaces* are now defined as follows. The invited speaker *InterestSpace* uses the same policy used in the last solution. The *Scopes parties* of all invited speakers and the microphone's *parties* are added to this *InterestSpace*. When an invited speaker wants to talk to the microphone, he must include it in his *Scope Expression*. Finally, the microphone *InterestSpace* enables everyone to have access to the information expressed by the invited speakers or audience members with microphone. This *InterestSpace* uses an "all-see-all" policy. The *parties* of all audience members and invited speakers and the *Scopes* of the microphone are inserted in this *InterestSpace*.

4.2.4.3 Brief discussion

This iteration shows the expressiveness power and the incremental development, qualities of the abstraction.

The expressiveness power quality is shown by transforming the microphone from a person's property into an entity described like the invited speakers or audience members. One could go to the extreme and represent everything like this, even the sound speakers or the conference table.

To implement this solution we can reuse all entities, apart from the microphone defined on the previous iteration, thus demonstrating the incremental development quality. The way of acquiring a microphone must change but the code responsible for this does not belong to the abstraction but to the domain logic inserted to deal with microphone ownership.

We would like to remark that by combining the *Aura* policy with the microphone entity one allows a user to be aware of other users that his aura alone would not enable him to be aware of the others. This emulates the third-party objects concept used in MASSIVE-2 [9] that affects the users by changing their values of aura, focus and nimbus.

4.2.5 Fifth iteration – More than one conference table

So far, we have considered that there is a single conference table. In this iteration we consider the existence of several conference tables, side by side, and a user can jump from one to another. A user from a conference table can see the topic that are being discussed in the conference tables adjacent his conference table he's in and, at any given moment, the user can leave one

conference to join another. A user cannot be in two conference tables at the same time.

4.2.5.1 Solution

To be able to deal with several conference tables, we have to compose policies to obtain the desired effect. Moreover, we need to create a way to represent the various conference tables and their interaction. The conference tables need to interact so that the topic of each table is forwared to its adjacent conference table. The solution of this iteration is as follows. First, we create a class, we call it *conference,* that contains a *Scope*, an *interestedParty*, a conference table, and that conference table's topic. We call instances of this class *locales*. The *Scope* of a *locale* express which locales are adjacent to it. The *interestedParty* represents the locale in the *awareness policy*. Second, we create an *InterestSpace* that manages the interaction between *locales* and satisfies the problem requirements. And encapsulate the conference table solution in the locales inserted in this new *InterestSpace*.

4.2.5.2 Implementation details

The implementation assumes each conference table is an object of *conference* type that implements the previous solution. Invited speakers and audience members maintain the same composition as described in the previous solution.

We define a new *InterestSpace*, called Locale *InterestSpace* that has a new kind of policy. This *InterestSpace* enables the interaction between the various *locales*. The *Scopes* and *interestedParties* of *locales* are inserted in this *InterestSpace*. To solve the interaction between conference tables, the *locale InterestSpace* use a new *awareness policy*. This policy consists in allowing the locales to see just their adjacent *locales*. The adjacent *locales* are expressed thought the *Scope*. When a *locale* is adjacent to another *locale* it sends its topic to the other *locale*. Whenever a user enters a locale, he is inserted in the *locale* conference table. A user can move to any of the adjacent *locales* of his locale. If the user decides to move, he is removed from the *locale* conference table and is inserted into the new *locale* conference table.

4.2.5.3 Brief discussion

This new policy, where the *Scopes* indicate who is adjacent to whom, is compatible to the SPLINE spatial regions and can even be more powerful since it allows each user to be aware of much more that just their immediate neighbours. For instance, if our locale *InterestSpace* implements an *Aura* policy, each locale would have, as adjacent, all the locales within his *interestedParty* range. If each locale has a different conference table awareness the example is comparable to the MASSIVE-3 adaptation of the SPLINE model.

The possibility to have *InterestSpaces* within *InterestSpaces* allows us even more expressiveness of our abstraction. This way we can have several spatial regions each of them with a different policy.

This example also shows that we can use the previous solution by composing policies which helps the development of a project.

5. EVALUATION

In this section we discuss what qualities we can be extracted from the various iterations.

Expressiveness power. Each iteration offers a different problem to solve and the abstraction is powerful enough to handle each problem in an efficient and elegant way thus effectively solving each iteration. The abstraction solves the basic problem and all the iteration with a minimum change from the basic solution. For instance, in the second iteration all we had to do was change the *InterestSpace* policy. Moreover we have shown that for the same problem, the abstraction can present several solutions with different kinds of advantages. We can then conclude that the abstraction has the proposed expressiveness power.

Incremental development. Starting at the basic problem and going through the various iterations, we can identify a strong similarity in the solutions. This means that the incremental development identified as a goal was achieved. We can use the abstraction to solve parts of the problem and then use incremental development to solve the rest of the problem. Each solution improves the previous one by replacing something on that solution or by adding something new. Since the problems are incremental, each solution builds on the previous one and does not need to remove any concepts that still apply. If there is a change of perspective (like in the "microphone-as-property" to microphone-as-entity change) the only code that changes is the one that relates not to the framework usage but with functional portions that need to change to cope with the new requirements. For instance, in the fourth iteration the invited speakers stopped owning a microphone since now the microphones are entities. This means that the abstractions are well defined and support the problem with accuracy. One example of this is the fifth iteration where the abstraction was composed to solve the problem. If the abstractions were not well defined that would not be possible.

Easy to use. The basic awareness management abstraction is composed of six concepts. Having so few concepts means that the framework is not hard to learn, use or even change. Nonetheless it has sufficient concepts that divide the problem into independent pieces of code which can be extended according to one's needs. For example if one wanted to hear or see further it would change his *InterestExpression* and if one would need to hear different languages one would add additional *InterestedParties*. To change the way two people interact, we need to change the *Awareness Policy*. All this allows the programmer to write less code and have simple

primitives to develop awareness solutions. For example to create a policy and *InterestSpace*, the policy is already defined, all we have to do is:

- policy = new AuraPolicy();
- space = new InterestSpace("name", policy);

By dividing the problem, into independent problems the framework has well defined spots of evolution. This means one can build a generic solution to a problem using the various concepts given by the framework and then fill in the gaps to achieve a specific solution. An example is given with the conference table where the generic solution consists of three *InterestSpaces* and one *Scope* and *Party* for each person. For each iteration of the conference problem we adapted that solution to solve the particular problem presented. Another question is why did we choose those solutions to the problems presented and not other possible solutions? To answer this question we use a real example. Consider the following situation: John is trying to say something to Mary but Mary is twenty meters away and does not seem to listen. What could we do in this example? Assuming we could do some absurd things in real life, we could do the following:

- Improve Mary's hearing so that she could hear John twenty meters away
- Change wave physics so that anyone could hear anyone. This would solve the problem even if John was one kilometer away
- Move Mary or John closer to one another so they could hear each other loud and clear
- Ask John to speak louder. This last solution seems the easier and best solution to the problem.

This problem can be solved by the framework as follows. John has a *Scope*, Mary has an *InterestedParty* and they are both inserted into an *InterestSpace* with an *Aura* policy. In our framework the above solutions correspond to:

- Increase the *interestExpression* range
- Modify the *Aura* policy to an "all-see-all" policy
- Move the *Scope* or *Party* so that their Expressions intersect
- Increase the *ScopeExpression* range

The solutions we chose in section four are those that make more sense. The framework is flexible enough to support with several solutions to the same problem. But as in real life, some of those solutions do not make sense. The solutions chosen are the most adaptable and the ones where the inserted code is more generic thus allowing more iterations to be done on top of them.

6. CONCLUSION

In this paper, we presented our solution to the awareness management problem. Our solution has the following qualities: expressiveness power, incremental development and easy to use. Our solution improves the time to design a solution since the framework is easy to use, the time to develop the basic solution since there is a small set of concepts to apply and even improves the reiterations (correct concepts, well defined spots to insert domain code and high code reusability). Due to the incremental development property, programmers can develop a basic solution and then build more complex solutions on top without wasting valuable resources. One problem our abstraction cannot solve is the fading of the information. For instance, when a police car, with its sirens on, starts moving away from a person, she should ear the sound fading away. It does not stop hearing the sirens just because the police car is too far. The current abstraction does not handle continuous expressions of information just discrete expressions that define when you can receive or not the information. This issue is being addressed by adding a filter abstraction. This seems to be a promising path of evolution for the framework. Filters are entities that transform information. Placed between *scopes* and *parties* they can be used to change the information being transmitted. The challenge here is maintaining the separation between information definition and information propagation, since a filter should be independent of how the information it filters gets propagated.

7. ACKNOWLEDGMENTS

This work was partially funded by Fundação para a Ciência e Tecnologia, Praxis/ C/ EEI/ 33127/ 1999 MOOSCo.

8. REFERENCES

[1] M. Antunes, A. R. Silva, and J. Martins. An Abstraction for Awareness Management in Collaborative Virtual Environments. In *Virtual Reality Software & Technology 2001(VRST 2001)*, Banff, Alberta, Canada, November 2001. ACM SIGGRAPH.

[2] M. Antunes, H. Miranda, A. R. Silva, L. Rodrigues, and J. Martins. Separating replication from distributed communication: Problems and solutions. In *International Workshop on Distributed Dynamic Multiservice Architectures (DDMA 2001)*, pages 103–108, Phoenix, Arizona, USA, April 2001. IEEE.

[3] M. Antunes and A. R. Silva. Using separation and composition of concerns to build multiuser virtual environments. In *6th International Workshop on Groupware (Criwg'2000)*, Portugal, October 2000. IEEE.

[4] J. Barrus, R. Waters, and D. Anderson. Locales: Supporting Large Multiuser Virtual Environments. In *IEEE CG and Applications*, November 1996.

[5] S. Benford and L. Fahln. A Spatial Model of Interaction in Large Virtual Environments. In *Proceedings ECSCW'93*, Milan, Italy, September 1993.

[6] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *In Proc. Computer Supported Cooperative Work (CSCW '92)*, 1992.

[7] T. Funkhouser. Ring: A Client-Server System for Multi-User Virtual Environments. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 85–92. ACM SIGGRAPH, March 1995.

[8] C. Greenhalg, J. Pubrick, and D. Snowdon. Inside MASSIVE-3: Flexible support for data consistency and world structuring. In *Proceedings of the 3rd International Conference on Collaborative Virtual Environments, (CVE'2000)*, San Francisco, California, USA, September 2000. ACM Press.

[9] C. Greenhalgh and S. Benford. Boundaries, Awareness and Interaction in Collaborative Virtual Environments. In *Proceedings of the 6th Workshop on Enabling Technologies (WET-ICE '97) Infrastructure for Collaborative Enterprise*, Cambridge, Massachusetts, USA, June 1997. IEEE Computer Society Press.

[10] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, and P. Barham. Exploiting Reality with Multicast Groups. In *IEEE CG and Applications*, September 1995.