

A FLEXIBLE DESIGN SOLUTION FOR REPLICATION IN COLLABORATIVE VIRTUAL ENVIRONMENTS

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

***Abstract:** Support for replication is an intrinsic characteristic of Collaborative Virtual Environments (CVEs) systems. A generic solution for replication must, not only be able to support application's different consistency requirements but also be sufficiently independent of distributed communication to allow it to coexist with different communication protocols and distributed architectures. To cope with these problems this paper describes an object-oriented design solution for the concern of replication in CVEs that separates replication from distribution and provides a tailorable solution for consistency maintenance.*

KEYWORDS: CVE, Replication, Object-Oriented framework, Separation of Concerns

1 INTRODUCTION

Collaborative Virtual Environments (CVEs) are networked virtual environments used to support collaborative work. Users are represented graphically within the environment and can perceive other users' actions through their graphical representation. Since CVEs are multi-user systems, all information about the virtual environment is shared among all users. Due to the need of real-time interaction, replication is used as a way to share the information among users. The environment is replicated (fully or partially) at each user's machine and it is necessary to guarantee the consistence of the replicated environment in the presence of the changes performed by the users. However, depending of the system's goals, different consistency requirements are needed. For instance, in a system where real-time interaction is the main goal, it is necessary that the changes performed on the environment be propagated to all the users as quick as possible, even if that implies relaxing the environment consistency. On the other hand, a system where collaboration is the primary goal, has stronger requirements regarding consistency. This means that a solution for replication in CVEs must be able to support different consistency requirements.

Another problem for supporting replication in CVEs is its dependency with the underlying distributed communication architecture. For replication to be supported, distributed communication will have to be considered. The problem is that different variations exist for supporting distributed communication. These variations exist both at the level of communication protocol and distributed architectures. Different communication protocols with different quality of service may be used depending of the types of data to be transmitted. Different distributed architectures can be used depending of the available network resources. For instance, it is possible to use distributed architectures like client-server, peer-to-peer and hybrid. Both architectures have their advantages and disadvantages thus, designers should be able to choose which is more suitable for the target application. Defining a replication solution tightly coupled with distributed communication may compromise the flexibility of that same solution. A generic solution for replication must not only be able to support application's different consistency requirements but must also be sufficiently independent of distributed communication to allow it to coexist with different communication protocols and distributed architectures.

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 [1]. 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].

This paper describes an object-oriented abstraction for the replication concern that is able to support different consistency criteria for state updates, different replication policies for creating and destroying replicas and handling newcomers, and is also independent of distributed communication. Actually, following a separation of concerns approach the proposed abstraction does not deal with any aspects of distributed communication. However, it was designed so that composition with distributed communication can be easily achieved, allowing different solutions for distributed communication to be supported.

This paper is structured as follows. Section 2 describes the related work. Section 3 presents the proposed replication abstraction. Finally, Section 4 presents some conclusions and future work.

2 RELATED WORK

All existing systems provide support for replication. However, they are committed to particular solutions for consistency maintenance and are dependent of the underlying distributed communication protocols and architectures. Furthermore, they allow for little or no customization regarding replication issues.

NPSNET [3] is distributed virtual environment for distributed military simulations. This system aims to support a large number of simulated users and entities. As a result, it provides weak consistency using dead reckon and state regeneration. Furthermore, it uses a peer-to-peer architecture with multicast protocols. This is a fully distributed architecture that does not depend of a single participant to enable a simulation.

In SPLINE [4, 5] the consistency is ensured by the ISTP (Interactive Sharing Transfer Protocol) protocol, an application level distributed communication protocol. The state update messages are supported by the ISTP protocol and contain the state update description instead of the actual value. This reduces the size of state update messages. Furthermore, the protocol can deal with message losses being able to obtain the new state even if a state update was lost. Another characteristic of this system is that it uses partial replication. The system is divided in disjoint regions called locales and each user only replicates a small subset of all existing locales. SPLINE uses a hybrid distributed architecture. Peer-to-peer is used for communication between users that replicate the same locale, and client-server is used for new clients to receive the initial state of a locale and to handle state updates losses. The partial replication improves system scalability by reducing the number of state updates that each user must receive.

MASSIVE3 [6] uses a distributed data base approach. An environment is constituted by several databases called *Environments*, each one describing a portion of the virtual environment. Each database contains any number of hierarchical structured data items that are the unit of replication. Each data item has an owner that is allowed to update or delete the item. Changes are performed to items in a database through event objects that describe the change and that are propagated to all the database replicas. The consistency maintenance supports both reliable and unreliable state updates. It also allows application programmers to define sequencers for event objects that impose ordering in the corresponding state updates. Finally, the system supports different consistency schemes: centralized updates can be used where all event objects for a given item are first handled by the item's owner and then propagated to the remain replicas; transfer ownership can be used so that the right to update a data item can be transferred between applications; and CIAO-style [7] updates can be used where the state update and request for transfer ownership are combined in one message. MASSIVE-3 uses a client server architecture implemented over the TCP protocol. Each database is managed by a server. Each server acts as communication router to propagate state updates between clients and also for delivering new clients the current state of the database. Partial replication is supported by using a mechanism similar to the one in SPLINE.

The approach described in this paper considers replication as an independent concern, in particular, independent from distributed communication. The replication abstraction deals exclusively with replication problems. Whichever aspects of replication dependent on other concerns, like distributed communication, must be dealt within separate composition modules. Any change necessary on those modules will only affect composition code and not replication code, thus assuring a greater level of reusability. Another important reason for considering replication as an independent concern is to allow the person responsible for the system development, or even adaptation according to the needs of certain

applications, to think on replication and the problems associated with maintaining consistency separately of the remaining concerns.

3 THE REPLICATION ABSTRACTION

This section describes an object-oriented abstraction for replication in CVEs. The goal of the proposed abstraction is to provide a common framework upon which different solutions for replication can be built. To be able to achieve this goal it is necessary that the abstraction is flexible enough to capture the several variations that exist in solutions for the problem of replication. Furthermore, the defined abstraction describes a solution for replication separately from distributed communication.

The description of the *Replication* abstraction is divided in three sections. Firstly, is described in a general form the problem that must be solved. Secondly, different variations that any flexible solution for replication should support are described. Finally, the solution is described, i.e. the proposed object-oriented abstraction for replication.

3.1 Problem

One of the decisions that have to be made when defining a solution for information sharing on CVEs systems is to define what is the information that must be shared and what must be the control of the programmer over what is shared information. For instance, should all objects of an application be shared, or even, which attributes or behaviors should be shared? The first approach allows hiding from programmers the details of information sharing but allows for little control over the shared data and doesn't take advantage of the semantics of objects used to define consistency criteria. The second approach makes explicit the existence of replication giving applications a better control over shared information management, although it has a price: programmers have to worry about replication related issues.

Mechanisms used to maintain shared information consistency are another major issue, namely which are the consistency criteria that must be guaranteed in order to maintain a consistent perception of shared information among all users. For instance, is it necessary for all users to see objects' state changes in the same order? When can state prediction algorithms be used and, if so, which should be used? Should the same consistency criteria be used for all objects, or different criteria may be defined for each object?

Finally, it is necessary to define the way new users receive the shared state of an application. Since different users may have created different portions of the shared state, who provides the shared state? Is there a single entity responsible for providing the shared state or is that responsibility split among several entities?

3.2 Variations

A solution for the replication concern should provide the following variations:

- **Shared state definition.** Shared state must be defined on a per-object basis. Each object must be able to define which attributes are to be shared between replicas. Only changes on shared attributes need to be propagated to the object's replicas. It should also be possible to define shared actions, object behaviors that are executed on every replica. Shared actions can be used to implement certain deterministic behaviors in all the replicas of certain shared objects.
- **Different consistency criteria support.** It is necessary to support different consistency criteria. These consistency criteria must be chosen on a per-object basis, or even based on a per-element of shared state of an object (an attribute or an action). It should be possible for applications to define new consistency criteria. Associating consistency criteria to shared state must have sufficient flexibility to allow the same criteria to be used for one or more attributes of the same object or different objects. For instance, forcing ordering between state updates of one or more attributes of the same object, or between attributes of different objects.
- **Different replication policies support.** Allow the definition of different policies for replica management, making it possible to deal with newcomers of a shared space in different ways. For

instance, policies may be defined where the participants obtain the shared state of a replication server that contains all the objects within the virtual world. On other occasions, it may be desired to have each participant responsible for managing the objects it creates and supplying replicas of those objects to newcomers.

3.3 Solution

The solution considers the replication of objects. The shared state of an object is defined explicitly as a subset of its attributes and actions (methods). Replication is the mechanism used to share information, that is, objects. Objects are shared in a particular shared space. A shared space is an abstract entity that represents a set of shared objects. Several shared spaces may co-exist, but each object is shared in only one space. It is necessary to become a member of a shared space in order to access its objects. The space members can insert and remove objects of the shared space. Objects inserted by a member are named local replicas. For each replica inserted by a member other replicas are created in the other members.

The proposed solution does not possess a concrete representation for shared space but for the view a specific member has of that space. This is due to the fact that a shared space is an abstract entity formed by the entire set of members of that space, more precisely, formed by the set of local replicas and replicas of all the objects inserted by the other members.

Consistency maintenance of a shared space is carried on two levels: at the object level and at the space level. At the object level, consistency maintenance seeks to guarantee that object state changes are propagated to all space members according to certain consistency criteria. Furthermore, a distinction is made between consistency criteria applied to state updates of a single object, intra-object consistency or of several objects, inter-object consistency. Consistency criteria are guaranteed by consistency protocols. Every intra-object consistency protocol is associated with an inter-object consistency protocol. All updates are first handled by an intra-object protocol that may force some intra-object consistency and then handled by an inter-object consistency protocol that forces inter-object consistency criteria.

At the space level, consistency maintenance aims at providing the space members with a consistent vision of the space. This means that all users are aware of the same set of shared objects, or that the set of objects that each user is aware of, or replicates, obeys to some replication criteria. This consistency maintenance is supported by a replication policy that manages the creation and destruction of replicas on space members, as objects are inserted or removed and members join or leave the space. The shared space membership and the creation and destruction of replicas is propagated to members of a shared space through a special consistency protocol, the space consistency protocol.

3.4 Implementation

The *Replication* abstraction described in this paper was implemented in JavaTM 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. For each concern an abstraction was defined and implemented. The support for CVEs was obtained composing each of the concern's implementations. At the present moment, there are abstractions for replication, awareness management, distributed communication and object interaction. By supporting different sets of concern's compositions it is possible to support CVE's with different capabilities, enabling in this way incremental development. For instance, scoped interaction is obtained by composing awareness management with object interaction. Adding replication and distributed communication, a distributed and replicated virtual environment is obtained. Partial replication of the environment is obtained composing replication and awareness management. Depending how the MOOSCo framework is instantiated (which composition the programmer chooses) the functionality of the CVEs will be different. 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.

In the remainder of this section we will present two applications developed using the MOOSCo framework and will describe their replication aspects.

3.4.1 Conference Table

This application consists of 3D chat distributed virtual environment. Users are represented in the environment through a 3D object called *Avatar*. Users can move within a 3D environment and can interact with each other by text-based communication. In the environment there is a conference table, around which users can sit. The table functionality only deals with managing the number of available sits, allowing users (or software agents) to sit at the table only if there is a free sit. If a user tries to sit at the table and there are no available sits the table sends him a text message saying that there are no available sits left for him.

The original goal of this application was to demonstrate the flexibility of the *Awareness Management* abstraction implemented in MOOSCo [8]. Awareness management is used to control the interaction between the users sat at the table and the remaining users of the environment. Users at the table cannot interact (chat) with users outside the table. Depending of the awareness configuration, users outside the table may or may not "hear" what the users in table are saying. Although Conference Table is a very simple application is can also be useful to show the flexibility and expressiveness of the *Replication* abstraction.

All users replicate the entire environment at their local machines. Replicas are created at each user site for all objects in the environment including other users' *avatars*.

This application uses two different intra-object consistency protocols and three different inter-object consistency protocols associated in different combinations. The inter-object consistency protocols, besides forcing inter-object consistency criteria to updates of different objects, also supports update propagation over the network. These inter-object protocols are in fact a composition between inter-object consistency protocol and distributed communication channels from *Distributed Communication* abstraction and support updates' propagation over a network, using a particular communication protocol with a particular quality of service and forcing a particular consistency criteria.

For text messages updates - which are in fact passed between objects (users) through the use of shared behaviors - no intra-object consistency criteria was needed, so a default intra-object consistency protocol was used that does not impose any kind of criteria, i.e., it applies the updates immediately on the correspondent objects. However, to allow users to follow the ongoing conversation more easily, it was replaced with an inter-object consistency protocol that forces causal ordering between user's messages. This causal ordering allows users to read the text messages in the correct order.

The third intra-object consistency protocol was used for maintaining, among the conference table replicas, the consistency of the number of available sits. This is necessary because different users are interacting with the table concurrently. Each user interacts with a replica of the table. In order to satisfy the table's functionality, i.e., a user can only sit at the table if there is an available sit, it is necessary to guarantee the consistency of the available site throughout the table's replicas. The replication micro-framework is flexible enough to support more than one solution for this problem. One possible solution is to differentiate between the conference table implementation and the conference table's replica implementation. In this case, the programmer could implement the conference table and the conference table replicas being aware of replication by using the replication framework facilities for exchanging messages between replicas of the same object and use those messages to force some kind of synchronization. This solution has the inconvenient of mixing the conference table functional code with replication specific issues and the code may become too complex. Nevertheless, this kind of solution can be useful at times. A simpler solution is, instead of making the available sits a shared attribute, to make the table's behaviors for a user to sit and to get up from the table shared. The result is that, when a user invokes that behavior on its table's replica that behavior is first propagated to the other replicas and then applied at the local replica. The act of sharing those behaviors does not solve the problem by itself; in fact it makes it worse since now it causes the behavior to execute multiple times, one at each replica. However, the use of shared behaviors in conjunction with an intra-object consistency protocol that forces the shared behavior to be only executed by a single replica, called the pilot replica, allows the problem to be solved. This intra-object consistency protocol, called *object ownership protocol*, is implemented by forwarding the updates from the replicas to the pilot replica. Since all shared behaviors are first processed

by the associated intra-object consistency protocols before being applied locally, the *object ownership protocol* guarantees that only the pilot replica executes the associated shared behaviors. In this way, all users' requests for sitting and standing up are serialized at the pilot replica, eliminating in this way any possible inconsistencies. For these behaviors, an inter-object consistency protocol is used that supports reliable non-ordered updates over the network. In this case, a reliable consistency criterion was used so that users could always see a result of their interactions with the table.

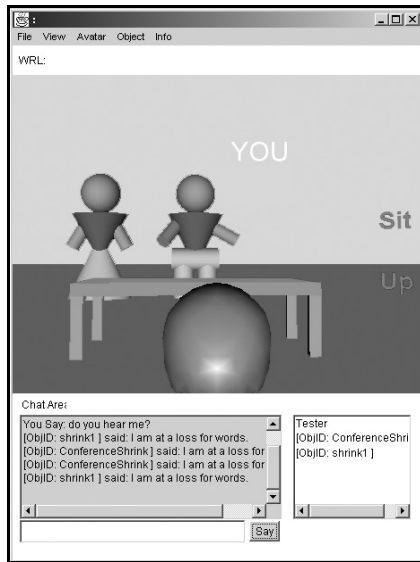


Figure 1 – Conference Table Application.

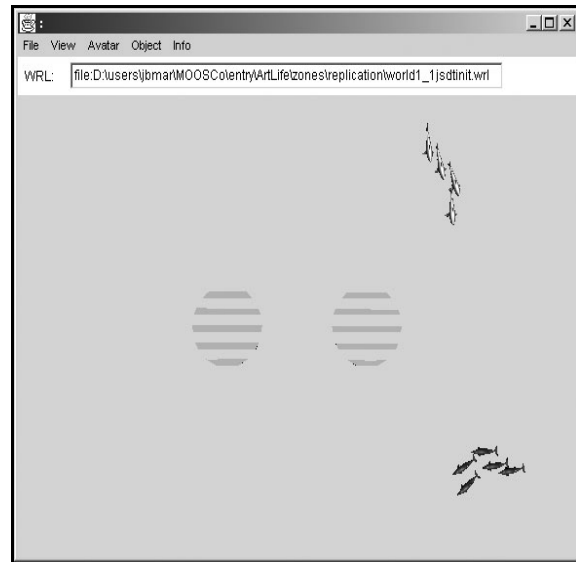


Figure 2 – Distributed Art/Life Application.

Note that not all MOOSCo applications need different consistency criteria; the consistency criteria are configured in a per-object and per-application basis.

Relative to the replication policy used, this application uses a *master-slave* replication policy. There is a replication server, the *master*, where the world is defined and its contents (the conference table). Every time a user, a *slave*, joins the world it receives the state of world from the master. Each slave also sends to the master the replica of its *avatar*. The master is responsible to create replicas of the new user's *avatar* in the existing users. Note that this master slave policy is independent of the distributed architecture. This application can run under a client-server distributed architecture (the replication master plays the role of server) or a peer-to-peer architecture using multicast protocols. The choice of which distributed architecture to use will depend of the available network resources and the target number of simultaneous users.

3.4.2 *Distributed ART/LIFE*

Distributed ART/LIFE is MOOSCo's adaptation of a previous standalone application. The original application was developed to demonstrate emergent behavior theories on aggregate moving. The application's main goal is to simulate a world inhabited by living creatures. Each creature reacts to its environments according to a set of predefined behaviors. These creatures may have one of two roles: predators or preys. The latter tend to aggregate in flocks, whereas predators will predominantly act alone. The *boids* model, used in ART/LIFE, was first defined by [9]. This model tries to describe a population's behavior through the individual behaviors of its elements. It was developed to describe the coordinated group movement of flocking birds, schools of fish or herds. To achieve the desired emergent behavior it relies on each individual's perception of its surroundings.

The Distributed ART/LIFE application supports a distributed simulation of the *boids* model in a 3D virtual environment. In this application each participant can simulate one or more *boids*. Each participant also replicates the other participant's *boids*. As a result, its *boids* perceive the other participants' *boids*

allowing the emergent behavior to develop. Since each boid's behavior is influenced by the other boids' behavior, to be able to support a distributed simulation and still observe the emergent behavior it is necessary to guarantee that the boids's replicas are as consistent as possible. Using ordering and reliable consistency protocols, theoretically, would give the necessary consistency. However, such consistency criteria would impose unnecessary delays to the replica updates causing erroneous emergent behavior, since each simulated *boid* would be influenced by old values for the position and orientation of the remaining boids' replicas. Instead of imposing strong consistency, a more relaxed consistency criterion was put in place that uses state prediction for the boids movement. For the boids' position attributes a dead reckon algorithm was used has an intra-object consistency protocol to predict position values of boids' replicas, and for reducing the number of updates sent over the network. For this attribute an inter-object consistency protocol is used that supports unreliable and non-ordered updates over the network.

In the original application the movement of the boids was limited to a well-defined region. This was merely to avoid *boid* dispersion, and to ease the emergent behavior observation. By using simple dead reckon algorithms, the boids replicas would sometimes cross the simulation bounds. Since the *Replication* abstraction allows the definition and customization of the consistency protocols, a dead-reckon protocol was defined that took into account the simulation spatial bounds. Actually, in the *Replication* abstraction's implementation, the intra and inter-object consistency protocols were implemented as stacks of protocol layers, allowing protocols to be constructed by stacking several protocol layers, each implementing a particular consistency criterion¹. The restriction to the simulation spatial bounds was implemented in a protocol layer that was placed on top of the dead-reckon protocol, thus allowing the use of the generic dead-reckon protocol and still force the limit restriction.

Different replication policies were used/defined for this application. The first policy is one of the default replication policies implemented in the MOOSCo application and is called the *distributed replication policy*. In this policy there is no central entity responsible for giving newcomers the current state of the environment, i.e., for creating replicas of the existing objects in the newcomers machine. This policy is ideal for supporting distributed simulation, since there is no single entity from which the simulation depends on. The second replication policy, *flock based replication policy*, is a variation of the first that supports partial replication. In this policy each *boid* is associated with a flock. Each simulation's participant indicates the flocks he is interesting in simulating and only replicates the boids belonging to those flocks. This policy allows the simulation of a greater number of boids since each participant only needs to replicate a subset of all existing boids. Of course if each flock is very populated then even with this policy each participant still needs to replicate a large number of boids. In that case other partial replication policies could be used for instance, the cell-based policy like the one used in [3]. Note that the partial replication policies are obtained by composition of *Replication* abstraction with *Awareness Management* abstraction also defined in MOOSCo [8].

4 CONCLUSIONS

In this paper an object-oriented abstraction for replication was proposed. This abstraction was developed under a separation of concerns approach for the developing of Collaborative Virtual Environments. In this approach, abstractions are defined for each of the different concerns of these systems. The described abstraction provides a generic solution for the problem of replication that is completely independent of distributed communication. It allows for explicit control over the definition of shared state. Also, it allows for different consistency criteria. One of the main characteristics of the defined abstraction is the distinction between consistency criteria applied to a single object (intra-object consistency) and consistency criteria applied to more than one object (inter-object consistency). Finally, the defined abstraction allows for different replication policies to be defined that manage the way newcomers are handled and replicas are created and destroyed in each shared space member.

¹The consistency protocols' implementation was based in several existing distributed communication platforms [10, 11] and the purpose was to promote protocol reuse and protocol composition

At present, abstractions were also defined for object interaction, distributed communication and awareness management concerns [2]. There is a Java™ implementation of the defined abstractions and their compositions. In this implementation, the replication abstraction was composed with the distributed communication abstraction to support distributed replicated virtual environments, and with awareness management and distributed communication abstractions to support partial distributed replication of virtual environments. As future work, different replication policies and consistency protocols will be defined and experience with them will help to further refine the replication abstraction. Moreover, other abstractions will be defined that support solutions for other concerns like persistence and authentication.

ACKNOWLEDGMENTS

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

REFERENCES

- [1] MOOSCo. Multi-user object oriented virtual environments with separation of concerns. MOOSCo Home Page URL: <http://www.esw.inesc.pt/moosco/>.
- [2] 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)*, Madeira, Portugal, October 2000. IEEE.
- [3] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, and P. Barham. Exploiting Reality with Multicast Groups. In *IEEE Computer Graphics and Applications*, pages 15(5):38–45, September 1995.
- [4] R. C. Waters, D. B. Anderson, J. W. Barrus et. all. Diamond park and spline: A social virtual reality system with 3d animation, spoken interaction, and runtime modifiability. In *PRESENCE: Teleoperators and Virtual Environments*, August 1997.
- [5] R. C. Waters, D. B. Anderson, and D. L. Schwenke. The interactive sharing transfer protocol version 1.0. Technical Report TR-97-10, MERL - A Mitsubishi Electrical Research Laboratory, 1996.
- [6] C. Greenhalgh, J. Purbrick, and D. Snowdon. Inside massive-3: flexible support for data consistency and world structuring. In *Proceedings of the third international conference on Collaborative virtual environments*, pages 119 – 127, MA USA, August 2000. ACM.
- [7] U.-J. Sung, J.-H. Yang, and K.-Y. Wonh. Concurrency control in ciao. In *Proceedings of the IEEE Virtual Reality Conference*, Houston, TX, March 13-17 1999. IEEE.
- [8] M. Antunes, A.R. Silva and J. Martins. An Abstraction for Awareness Management in Collaborative Virtual Environments. Submitted to *ACM Symposium on Virtual Reality Software & Technology 2001*.
- [9] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Proceedings of the SIGGRAPH '87 Conference*, pages 21(4):25–34, Anaheim, California, July 1987. ACM.
- [10] H. Miranda and L. Rodrigues. Flexible communication support for CSCW applications. In *5th International Workshop on Groupware - CRIWG'99*, pages 338–342, Cancún, México, September 1999. IEEE.
- [11] M. Hayden. The Ensemble System. PhD thesis, Cornell University, Computer Science Department, 1998.