

Using Separation and Composition of Concerns to Build Multiuser Virtual Environments

Miguel Antunes *and António Rito Silva
INESC/IST Technical University of Lisbon
Rua Alves Redol n^o9, 1000-029 Lisboa, PORTUGAL
{Miguel.Antunes, Rito.Silva}@inesc.pt

Abstract

Developing Multiuser Virtual Environments (MUVE) is a very complex task since it involves several engineering domains aspects such as, Virtual Reality, Cooperative Work and Distributed Systems. Furthermore, existing technologies like, graphics capabilities, processing power and network bandwidth, are permanently evolving. For MUVE systems to be successfully developed they must not only deal with all the different aspects that are inherent to these systems but also have the ability to deal with the continuous technology and requirements evolution. To cope with these problems a software engineering separation of concerns approach is proposed. Concerns are identified for each of the different aspects of MUVES. To obtain the necessary flexibility, each one of the concerns should abstract its possible variations in order to support the future system evolution. Moreover, it is shown how concerns composition has the necessary expressive power to build Multiuser Virtual Environments.

1. Introduction

For several years a number of middleware systems for the development of Multiuser Virtual Environments (MUVE) have been developed. These systems allow the MUVE developer to focus on the application's specific functionality instead of having to deal with the details and complexity of programming distributed and concurrent applications.

There are three main factors that make the developing of MUVE systems a highly complex task.

- *Inherent complexity of MUVE systems.* Developing Multiuser Virtual Environments is a very complex task since it involves several different engineering domains

aspects such as, Virtual Reality, Cooperative Work and Distributed Systems.

- *Different Requirements.* The existence of very different requirements means that there must exist different solutions for the same aspect. For instance, a MUVE more focused in collaborative work has different requirements regarding consistency than a MUVE more focused in social interaction. Depending on the network capabilities of the intended users and on scalability requirements, different solutions may be used for dealing with the distribution aspects of the system.
- *Technology evolution.* Existing technologies like graphics capabilities, processing power and network bandwidth, are permanently evolving. As a result particular solutions that are suitable for today's technology may not be suitable tomorrow.

For MUVES systems to be successfully developed they must not only deal with all the different aspects that are inherent to these systems, but also be able to deal with the continuous technology and requirements evolution.

To cope with these problems a software engineering separation of concerns approach is proposed. Separation of concerns is considered a key technique in Software Engineering [10]. This technique consists of treating different aspects of program construction separately in order to control complexity in accordance with a divide-and-conquer approach.

DASCo [5] is an approach defined for concurrent and distributed programs, which separately handles, during the development process, functional and non-functional concerns. This approach defines solutions for each non-functional concern related to concurrency and distribution. In addition, the approach allows the most appropriate solution for each concern to be customized by the programmer and to take into account the specific needs of the program being built. Concern solutions are based on micro-architectures of co-operating objects, which may be applied

*This work was funded by Fundação para a Ciência e Tecnologia

and customized in various programs.

This paper describes how the separation of concerns approach to the development of MUVES middleware can improve the overall quality of the implemented systems. It applies the DASCo approach to the development of Multiuser Virtual Environments. For each of the different aspects of MUVES, different concerns are identified. To obtain the necessary flexibility, each one of the concerns abstracts its possible variations in order to support the future system evolution. Moreover, concerns compositions are realized in order to obtain the necessary expressive power to build Multiuser Virtual Environments, while still maintaining the concerns variations.

The rest of this paper is structured as follows. Section 2 describes in detail the motivation and the problems with current approaches. Section 3 describes the several concerns that will be considered. Section 4 discusses those concerns' composition. Section 5 presents the conclusions.

2. Motivation

As in other application domains, in the MUVES domain there is not a unique and best solution. Solutions should be contextual. For instance, the complete satisfaction of requirements for consistency, adaptability, scalability and efficiency is not easy and may result in conflicting and inconsistent solutions. In addition, domain-specific requirements may consider different levels of consistency and even their change at runtime.

MUVE applications allow geographically distributed users to interact with each other and to manipulate the objects that exist in that the environment. For these systems, the sharing of information is one of the most important aspects. The way the information is shared and how the consistency of the shared information is enforced, will have a tremendous impact in the system's usability and scalability.

All the existing systems provide support for information sharing. DIVE [6] and SPLINE [1] uses a replicated database approach. All user's and application's interaction is done through the replicated database. Although they offer a clean separation between the application and the replicated database, applications have little control over the replication issues. For instance, dead-reckon algorithms are used to reduce position updates. Nevertheless, none of these systems allow the application developer to specify customized algorithms. Furthermore, they use reliable and ordered communication protocols for state updates. However, for some specific applications, unordered or even unreliable protocols could be sufficient.

Since MUVE applications are distributed applications, the choice of the distributed architecture to be used is also of extreme importance. From the existing systems and literature, the distributed architecture of these systems can be

categorized in: pure ClientServer (NetEffect [4], RING [7]), pure Peer-to-Peer (MASSIVE-2 [8], DIVE, NPSNET [9]) and Hybrid (GMD [3], SPLINE). All of these architectures have some advantages and some drawbacks.

- Client-Server architectures are simpler to manage. Servers offer an opportunity for smart message filtering and aggregation in order to reduce network bandwidth. A drawback of these architectures is that servers are also the cause of latency and scalability limits and can become severe bottlenecks. Despite that, there are situations where client-server architectures are the only possible choice. This is the case of Internet based MUVES where multicast support is not yet widely available.
- Peer-to-Peer architectures have the advantages of minimizing the latency for message delivery by sending messages directly to the receivers. Moreover, with peer-to-peer systems there is no central point of failure or central source of latency, which allows these systems to be more scalable. However, they consume potentially more bandwidth since they offer fewer opportunities for message filtering.
- Hybrid architectures try to capture the benefits of both architectures, client-server and peer-to-peer by using peer-to-peer communication between servers for lower latency and client-server for connecting to end-users for message filtering. However, these are the most complex ones and harder to manage, since besides keeping consistency between the clients, coordination between servers is also necessary .

Awareness in MUVE context defines the information of the environment that a process has access to. The information includes object updates, communication (text, audio and video) and information about other users (it may be simply visual information that allows to see the other users location and actions). Most of the existing systems use awareness management as a mechanism for reducing network bandwidth and increasing scalability. MASSIVE-2 extends the awareness management to promote user collaboration by using it to scope user interaction [2]. Users may have different levels of awareness of each other. Higher awareness levels correspond to higher interaction levels, e.g., being able to hear the other user, while lower awareness levels correspond to lower interaction levels, e.g., being aware that the other user is talking but not being able to hear him.

Several awareness management policies have been used.

- In RING and GMD 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 open space environments.

- **SPLINE** partitions the environment in spatial regions called locales. Each user is aware of all the objects in the current locale and the immediate neighbors. The partitioning of the environment is a very 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. There are some situations where different policies could be more useful. For instance, if an opaque wall separates two locales, and there is no direct access from one to the other, there is no point for a user in one of the locales to receive information about the objects in the other locale. In [11] an extension to the Spline is described where different policies may be used for defining what locales a user should be aware of at each moment.
- **MASSIVE-2** and **NPSNET** use spatial aura approaches. In **MASSIVE-2** each user is only aware of the objects that are in the range of its aura. **NPSNET** divides the environment in regions, and users are only aware of those regions, which their aura intersects with.

Existing systems have been successful in supporting MUVE applications. Most of them have focused in a particular set of requirements, e.g., scalability, collaboration and end users network capabilities. Few systems, however, provide enough flexibility to allow specific application customization and evolution. From a software engineering perspective, they fail to provide a fully-fledged compositional, reusable and customizable approach to the design and implementation of MUVEs.

With new commercial systems appearing every year, technology and user requirements constantly evolving, flexible support for evolution is of extreme importance. The failure of existing systems to support evolution is mainly due to the lack of proper abstractions and very strong dependencies between certain system aspects, for instance, information sharing and distributed architecture. As a result of their monolithic structure these systems are restricted to a limited set of middleware solutions.

3. Separation of Concerns

Concerns define individual aspects that can be dealt with separately from other aspects. Each concern allows the common abstractions and collaborations to be captured, and the variations to be identified and isolated.

Variations are a very important issue in concerns. If problems with multiple variations are not considered separately then there is a high risk that the resulting solution will be tightly coupled with other system aspects. This coupling will compromise the use of different variations and thus flexibility to deal with all the dimensions of the problem. Furthermore, changing the particular solution may imply changing different parts of the system.

In some circumstances only a particular variation is useful. Nevertheless, even in these situations a separation of concerns approach should be used since systems requirements evolve, a good solution for today may not be for tomorrow.

When considering MUVEs applications several concerns must be dealt with, for example: object interaction; awareness management; distributed communication; information sharing; concurrency control and authentication.

The above list is not an exhaustive one, only the main concerns were mentioned, but it is sufficient to show why MUVE applications are so complex to develop.

Initially, the work described in this paper will focus only in the first four concerns. They were chosen because all of the existing systems deal with them in one way or another. All other concerns will be treated and introduced in future work.

Another advantage of the separation of concerns approach is that allows for incremental development. Initially only a few concerns are considered. After proper compositions have been created, new concerns may be developed and further compositions may be obtained, augmenting the system functionality in an incremental way.

Object Interaction. The object interaction concern deals with object interaction inside an environment. A MUVE is populated with several objects, some controlled by human users, e.g., avatars, while others are autonomous. All objects may interact and communicate with the other existing objects.

This concern tries to capture the necessary abstractions for inter-object communication. It considers a MUVE as a sea of loosely coupled interacting objects. These abstractions must take into account specific requirements of MUVE systems. For instance, the possibility of an object to interact with a group of objects at the same time, i.e., there must be abstractions that support some form of group communication.

Note that the inter-object communication mechanisms defined in this concern are completely independent of any form of distributed communication that may be used.

The variations supported by this concern are the multiplicity of the interactions, one-to-one or one-to-many, and the type of the communication, message based, stream based for passing stream data between objects, synchronous

or asynchronous.

An abstraction to support the object interaction concern should consider the following elements:

- **Object**. Defines the abstraction for the MUVE objects. MUVE objects have an identity and have the ability to interact with other objects.
- **Environment**. Defines the execution environment that objects inhabit. Objects may enter and leave the environment. The environment allows objects to discover each other.
- **Inter-Object Connection**. Defines the abstractions that allow one object to interact with other objects. The object connections are divided in input connections, that allow one object to interact with a single object, and output connections, that allow one object to interact with one or more objects that are registered with that connection. Objects can use the environment to create and obtain the object connections of the objects they intend to interact with.

An example of single-user interaction is a user picking an object by sending it a *pick* message through the object's *IPickable* input connection.

An example of one-to-many interactions communication is: a user talking with other users using its audio output connection that is a stream based connection.

Awareness Management. This concern deals with the awareness management issues, i.e., what kind of information may be of interest and to whom is that information relevant? How do objects represent interest in information, and how those interested parties are managed so that they obtain only the information they are interested in?

Awareness management is considered a separate concern because it is an aspect that can have different solutions. Furthermore, particular solutions may be targeted to different goals. For instance, awareness management can be used to control what information is sent to each user over the network, and thus improving system scalability. Awareness management can also be used to scope inter-object interaction according to application specific rules. Although this will not increase system scalability by itself, it can be an useful mechanism for improving collaboration.

This concern does not make any assumptions about what should be used as awareness information. Its goal is to define the abstractions that allows the management of which objects should receive the awareness information. The variations supported by this concern are the awareness information, interest expressions and scope expressions representation, and different awareness management policies. Different policies have been used in the existing systems: visi-

bility based policies, spatial proximity policies or policies based on semantic information.

An abstraction to support the awareness management concern should consider the following elements:

- **Awareness Expressions**. There are two types of awareness expressions: Interest Expressions and Scope Expressions. The first type describes the interests of an interested party. They are used to decide what scopes a party should join. The second type describes the information that a scope provides. Both types of awareness expressions may be dynamic, i.e., they may change over time. Those changes will cause changes in the scopes' membership.
- **Scope**. Represents a context of awareness information. Interested parties can join and leave scopes accordingly to their interests. Examples of scopes may be: an object willing to interact with others or a room where all users in it can talk with each other. Each concrete scope defines what is the meaning of joining and leaving a scope. For instance, the result of a user joining a scope defining a room is that it enables him to communicate with the other users in the room.
- **Interested Party**. Represents any entity that may be interested in joining or leaving scopes.
- **Interest Space**. Defines a container for scopes. Parties register in interest spaces, by specifying interest expressions. Interest spaces may be nested.
- **Membership Strategy**. Represents the awareness policy. Each strategy has the responsibility of deciding what scope should an interested party join or leave. This is accomplished by matching, the party's interest expressions and the scope's expressions in a policy dependent way.

Using the elements defined above, an aura based awareness management strategy can easily be defined. Both interest expressions and scope expressions can be defined as spatial auras. The membership strategy will play the role of an aura collision manager. When a collision is detected between an interest expression's aura and a scope expression's aura, the associated interested party will be informed to join the correspondent scope. When the aura collision ceases the interested party will be informed to leave the scope.

Note that the strategy described above is completely independent of interested party and scope definition. Different interested parties and scopes can be used for the same awareness policies. They define the behaviour associated with leaving and joining scopes. For instance, a user A joining user B's audio scope will enable A to hear what B is saying. An object X joining object Y interaction scope

will mean that Y will be able to interact with X since it has entered Y's interaction range.

Distributed Communication. This concern deals with the distributed communication aspect of MUVES. It defines the abstractions for supporting the communication between all the distributed participant processes and different distributed architectures.

Distributed communication is of extreme importance in MUVES systems. First, because MUVES are inherently distributed. Second, because all the decisions regarding distributed communication may have a dramatic impact in the system performance, scalability, and the ability of supporting application requirements.

Given the different possible solutions and implementations for distributed communication, it is important to consider it separately from other system aspects. First, it allows focusing on the different communication requirements and correspondent set of possible solutions that best fit the MUVES domain. Second, it allows further tailoring of communication considering the individual specific needs of other system aspects as transmitting state updates, special data such as sound or video or any kind of control information that must be passed between systems participants.

Variations of this concern are: particular network protocol implementations, e.g., unicast and multicast; different quality of services, e.g., reliable, unreliable and ordered; and the different distributed architectures, e.g., client-server, peer-peer and hybrid. Support for different kinds of optimizations is also considered. Optimizations may be domain independent like message compression, or may take into account domain specific semantics, such as message aggregation.

An abstraction to support the distributed communication concern should consider the following elements:

- **Channel.** Represents a network communication channel. There can be unicast channels or multicast channels. The distinction between the types of the channels is based on the communication semantics, not on specific network protocols. Unicast channels allow one-to-one communication while multicast channels have group communication semantics.
- **Member.** An entity that is associated to a particular channel for receiving or reading messages. The same entity can be member of more than one channel. A unicast channel only has two members while multicast channels may have any number of members.
- **Message.** Application level messages that are changed between members through channels.

In a client-server architecture, both unicast and multicast channels can be implemented using a server as a mes-

sage router. Servers may apply message compression and aggregation to reduce network communication with clients as much as possible. Channel membership is used for the server to know which messages should be sent to each client.

In peer-to-peer architectures unicast channels may use direct point-to-point protocols, e.g., UDP and TCP, and multicast channels may use multicast protocols, e.g., IP Multicast, or any multicast protocol built on top of it.

Since applications only know unicast and multicast channels and not their particular implementations, it is possible to change their underlying distributed architecture without having to change the applications.

Information Sharing. The information sharing concern deals with the problem of creating and maintaining consistency among different replicas of the same object. More precisely it deals with the following questions:

- **Shared Information Definition.** What should be the shared information level of abstraction? Should it be plain data, attributes, or should high level events also be shared, so that shared behaviors may be supported? What should an object's shared state be?
- **Consistency Criteria.** What are the consistency criteria for state updates? Should all state updates be ordered and delivered to all replicas, or are there some state updates that do not need to be ordered and that can even be suppressed in some situations? In what circumstances can state prediction algorithms be used and what should those algorithms be?
- **Information Sharing Policy.** What kind of replication should be used: active replication, passive replication, or a combination of both?
- **Replica Life Cycle Management.** When should object replicas be created? When should they be destroyed? Who has the responsibility of creating object replicas for the newcomers? Should that responsibility belong to a single entity or should it be distributed by every entity that has created a shared object?

Variations of this concern are: the ability for each object to define precisely what is its shared state; support for different consistency criteria, and for associating different criteria to shared states in per object or even per attribute basis; support for different information sharing policies, active versus passive replication; and allow for different policies for handling newcomers.

The information sharing concern defines the following elements:

- **Shared Space**. Defines a space of shared objects. Objects may be added and removed from shared spaces. Every object added to a shared space will be replicated in all processes that have joined the space. Processes may join and leave shared spaces.
- **Space Member**. A process that has joined a shared space. A process may be member of several shared spaces at the same time. Every member is uniquely identified. A member may own objects. When a member leaves a shared space every object owned by him will be removed from the shared space.
- **Shared Object**. An object that has been added to a shared space. For simplicity, it is considered that an object can only belong to a shared space at a time.
- **Shared Object State**. Defines what is the shared state of an object. This allows for a clean separation of what parts of the object state should be shared and what parts shouldn't be shared.
- **Replica Manager**. The entity responsible for managing a local replica of a shared object and maintaining it consistent with the other replicas. Replica Managers are responsible for, among other things, knowing how and when to create and destroy replica instances, to dispatch update messages to the respective object and to define the consistency criteria of the object shared state. Behaviours are shared using object's Replica Managers.
- **Consistency Criteria**. Each consistency criteria represents certain consistency requirements. Each object's replica manager defines one or more consistency criteria and associates it with the object's shared state. State prediction algorithms may be part of consistency criteria.

Users are represented by avatars in the environment. Each user is able to see other user avatars. The user location is part of the shared state of an avatar so that users can see other's locations at all times. User movement can be predicted to a certain degree, by using user's velocity, current position and orientation. A dead-reckon algorithm is used to reduce the number of position updates. Since position updates can only be caused by the avatar controller, the user, the ordering of position updates can be easily implemented by the dead-reckon algorithm.

4. Composition of Concerns

In this section the different concern compositions are described. Compositions must be constructed in ways that allow the characteristics and variability's of each of the intervenients concerns to be maintained.

For each composition a description of its purpose, the constraints that result from it and how they may be enforced is presented.

In this paper only some of the compositions are described due to space limitations.

Note that in the MUVE applications context, some of those compositions may not make sense to be considered individually. Nevertheless, they will be considered because, from a software engineering perspective, it may be useful to support those compositions in order to allow an incremental approach to the development of MUVE applications. In such an approach a developer would start with simple compositions and change to more complex ones in an incremental way. 5and defining awareness management issues, and afterwards focus in

Scoped Interaction. The composition of object interaction and awareness management supports scoped interaction. Instead of all `Objects` being able to interact with each other in a given `Environment`, they will be able to interact with only a subset of the existing `Objects`.

The elements defined by this composition are:

- **Interaction Scope**. A specialization of `Scope` that defines a context where object interaction can occur, i.e., a context where `Objects` can obtain knowledge about other `Objects` and their `Inter-Object Connections`. For instance, interaction scopes can be defined by every object that is willing to interact with other objects. Alternatively, an interaction scope can be defined to allow interaction between all objects that have joined it. This may be used to implement something similar to third party objects, a spatial mode of interaction extension, defined in MASSIVE-2.
- **Interaction Party**. Extends `Interested Party` to represent every object that is available for other objects to interact with it, i.e., every object that has input connections. When an `Interaction Party` joins an `Interaction Scope`, it enters the interaction range of the `Object` that has defined the scope, which means that it may receive messages from that `Object`.
- **Scoped Environment**. An `Environment` that has one or more `Interest Spaces` associated with it. Allows objects to register as interaction parties, interaction scopes or both. A scoped environment may define only one interest space or it may define several interest spaces, one for each type of interaction such as audio interaction, chat interaction and object manipulation.

The composition maintains the variations from both concerns. `Inter-Object Connections` may still be of different types, one-to-one or one-to-many and message based or stream based. `Concrete Interest Expressions`, `Scope Expressions` and `Membership Strategies` may be defined in order to implement different awareness management policies.

This composition when considered individually will not be very useful for developing MUVE applications, since it does not have any distribution. However, it may be very useful when considering an incremental approach to the development of MUVES. Developers may develop and test new objects or even new awareness management policies in a single user configuration, totally ignoring distribution issues. Once implementation is completed and tested, then and only then, the developer must worry about other system aspects like distribution communication or replication.

Distributed Replicated Objects. The composition of the distributed communication and replication concerns provides support for replicating MUVE objects across different processes.

The following elements are defined in this composition:

- **Distributed Consistency Protocol.** It represents a distributed protocol that enforces a particular `Consistency Criteria` so that shared state can be maintained consistent among all distributed replicas. Each particular protocol will be implemented using a multicast `Channel` with the proper quality of service in order to support a particular `Consistency Criteria`. For instance, strong consistency protocols may use `Channels` with reliable, and total order message transmission. Weak consistency protocols may use reliable or even unreliable `Channels`.
- **Distributed Replica Manager.** This is an extension to `Replica Manager` so that every state update and shared behavior events are sent to an object shared state through one or more distributed consistency protocols.
- **Distributed Membership Protocol.** Represents a distributed protocol for managing `Space Members`. It uses the `Member` concept from the distributed communication concern, to manage space members as they join and leave a shared space. It coordinates with `Distributed Replica Managers` to transmit the shared state of the replicated objects, to every new member that has joined the shared space according to a particular newcomers policy.

Variations of this composition are: the distributed architecture of the shared space; support for different consistency protocols that take into account the particular distributed architecture, available communication protocols; end users network capabilities; and shared object's state consistency criteria.

As in the distributed communication concern, client-server architectures may be implemented by using client-server `Channel` implementations. However, in this case besides the server acting as a message server, through which all update messages are sent, it will also act as a replication server, from which every new space member will get the current state of all shared objects.

Peer-to-peer architectures may be implemented using peer-to-peer `Channel` implementations such that state updates are transmitted directly to all `Space Members` and where the responsibility to handle newcomers is distributed throughout all existing members.

Note that this composition does not support any kind of inter-object interaction. This means that this composition is only useful for objects that do not interact with other objects. For instance, it can be used for objects that are part of the world structure such as buildings and furniture. Note that this objects can also have behaviour. For instance, an object that simulates a growing plant does not interact with other objects and still needs to have its state changes propagated to all its replicas.

Partial Replication with Scoped Interaction This is the most complex of the compositions. It composes all concerns described in this paper. Besides providing support for replicated objects and scoped interaction, partial replication support is also provided.

By adding awareness management to replication and distributed communication, it is possible to obtain partial replication support. Partial replication means that only a subset of the existing objects will be replicated at any user process at each moment. The set of replicated objects is dynamic, i.e., it changes as a result of users actions, for instance, by moving around the environment.

The purpose of partial replication is to minimize the amount of objects that must be replicated in every user process at a given moment and so reducing the number of update messages that must be sent to each user process.

Besides the elements defined in the previous compositions the following new elements are defined for supporting partial replication:

- **Replication Interested Party.** A specialization of `Interested Party` for representing processes, that intend to create local replicas for some of the existing replicated objects. Usually this will represent user processes.

- **Replication Scope.** A specialization of Scope that defines collections of replicated objects. Joining and leaving replication scopes has the effect of creating or destroying local replicas of all the scope's objects, respectively.
- **Replication Space.** Defines an Interest Space of Replication Scopes. It composes Interest Space, Shared Space and Scoped Environments. In this way each user process's Environment will only contain a subset of all existing objects. It defines the user's view of the environment.

One possible solution for implementing replication scopes is to associate a different shared space to each replication scope. This solution has the advantage that each interaction scope can have its own underlying distributed architecture. However, it is necessary to guarantee that all interaction scopes provide the same consistency criteria so that objects can move between interaction scopes and still have their consistency requirements fulfilled.

As described in the awareness management concern, different awareness management policies may be achieved by defining special purpose Interest Expressions, Scope Expressions and Membership Strategies, e.g., spatial aura based policies. In partial replication, different awareness management policies correspond to different partial replication strategies.

Because awareness management was considered as a separate concern it is possible to use particular awareness management policies for different purposes. For instance it is possible to use a spatial aura membership strategy for both scoped interaction and partial replication. Although the same policy may be used for both situations the overall result is different, because different specializations of Scope and Interested Party are being used.

This composition allows the developing of MUVE applications with support for inter-object communication, different distributed architectures and different replication policies. Also it supports different awareness management policies for both scoped interaction and partial replication.

5. Conclusions

This paper presents work done in the context of MOOSCo (Multiuser Object-Oriented Virtual Environments with Separation of Concerns). MOOSCo proposes a software engineering separation of concerns approach. For each of the different aspects of MUVEs, different concerns are identified. To obtain the necessary flexibility each one of the concerns abstracts its possible variations, in order to support the future system evolution. Moreover, it describes how concerns composition have the necessary ex-

pressive power to build Multiuser Virtual Environments and how they can be integrated in an incremental development process.

This paper describes a first step to define the main abstractions that are part of a MUVE system. There is much to be gained by identifying those abstractions. They allow reuse and the establishment of a solid knowledge base that can be further extended and reasoned about.

In the future, additional concerns will be considered and composed with existing ones, as persistence, security and concurrency control.

References

- [1] J. Barrus, R. Waters, and D. Anderson. Locales: Supporting Large Multiuser Virtual Environments. In *IEEE Computer Graphics and Applications*, pages 16(6):50–100, November 1996.
- [2] S. Benford and L. Fahlén. A Spatial Model of Interaction in Large Virtual Environments. In *Proceedings ECSCW'93*, pages 13–17, Milan, Italy, September 1993.
- [3] W. Broll. Distributed Virtual Reality for Everyone - a Framework for Network VR on the Internet. In *Proceedings of the IEEE Virtual Reality Annual Symposium 1997 (VRAIS'97)*. IEEE Computer Society Press, 1997.
- [4] T. Das, G. Singh, A. Mitchell, P. S. Humar, and K. McGree. NetEffect: A Network Architecture for Large-Scale Multi-User Virtual Worlds. In *Proceedings of the IEEE Virtual Reality Annual Symposium 1997 (VRAIS'97)*. IEEE Computer Society Press, 1997.
- [5] DASCo. Development of Distributed Applications with Separation of Concerns Project. DASCo Home Page URL: <http://www.esw.inesc.pt/~ars/dasco>.
- [6] E. Frécon and M. Stenius. Dive: A Scalable Network Architecture for Distributed Virtual Environments. In *Distributed systems Engineering Journal (Special Issue on Distributed Virtual Environments)*, number Vol. 5, No 3, pages 91–100, September 1998.
- [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. Greenhalgh and S. Benford. Boundaries, Awareness and Interaction in Collaborative Virtual Environments. In *Proceedings WETICE'97*, pages 18–20, Cambridge, Massachusetts, USA, June 1997.
- [9] 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.
- [10] D. Parnas. Some Software Engineering Principles. In *Structured Analysis and Design: State of the Art Report*, pages 237–247. INFOTECH International, 1978.
- [11] J. Pubrick and C. Greenhalg. Extending Locales: Awareness Management in MASSIVE-3. In URL:<http://www.crg.cs.nott.ac.uk/research/systems/MASSIVE-3>, September 1999.