Virtual Worlds
From Concepts to a Distributed Implementation Framework

Patrik Fuhrer and Jacques Pasquier-Rocha

University of Fribourg
Department of Informatics
Rue P.-A. de Faucigny 2
CH-1700 Fribourg
Switzerland
patrik.fuhrer@unifr.ch
WWW home page: http://diuf.unifr.ch/people/fuhrer/

Abstract. This paper provides an extensive study, from definitions and concepts to a concrete extensible object-oriented software framework, for the challenging application domain represented by virtual worlds. Traditional solutions are based on centralized architectures and do not scale well. MaD-ViWorld, the prototypal software framework already implemented using Java and RMI by our group, allows for distributing the subspaces (rooms) of a given world on an arbitrarily large number of machines, each running a small server application. This very decentralized implementation strategy, as well as the simple and systematic way provided in order to populate the world with new types of active objects, represent the main originality of our approach.

Keywords: Virtual World, Framework, Distributed Computing, Distributed Events

Contents

1 Introduction

Since several years, our research group is involved with the task of designing and implementing distributed object-oriented virtual worlds. The aim of this paper is to present MaDViWorld [1], our prototypal extensible software framework supporting the creation of such shared environments. In order to best achieve this goal, it is organized as follows. The introductory section positions our work within the vast research field encompassed under the term *Virtual Worlds*. Section 2 describes a simple typical scenario illustrating the use of our software framework in order to precise the terminology and for introducing MaDViWorld underlying conceptual model. Section 3 presents an in depth discussion of the software solution that has been chosen in order to implement the model. Finally, the main achievements of our work are summarized and an insight is provided into what directions further research projects are currently investigating.

1.1 The Virtual World Paradigm

We have become accustomed to seeing the Cyberspace as a great sea of World Wide Web documents. This classic approach corresponds to the *document paradigm*. More fantastic views of Cyberspace portray it as a labyrinth of interconnected virtual worlds inhabited in real time by millions of people represented as "avatars" [2] and being aware of each other presence and actions. This vision corresponds to the *virtual world paradigm* and moves the Cyberspace beyond simple linked web documents, allowing for a much richer interaction between its users. The interested reader is referred to [9] for a more in depth presentation of the advantages of the latter paradigm.

1.2 About Virtual Worlds

Heudin [5] introduces Virtual Worlds as a new field of research that studies complexity by attempting to synthesize digital universes on computers. This includes models of simple abstract worlds such as cellular automata to more sophisticated virtual environments using Virtual Reality and Artificial Life techniques. Moreover, Bruce Damer, in the fifth chapter of [5], introduces the concept of Virtual Community.
Virtual Reality is concerned with the design of graphical spaces using advanced three-dimensional image synthesis. Besides displaying 3D spaces, two other important aspects are involved: immersion and interaction. The operator typically evolves in the generated world thanks to data suit, head mounted display and data gloves. The idea is to feel "physically" present in the virtual environment and to interact with it.
Artificial Life is focused on the simulation of living systems, and addresses the study of complex phenomena such as self-organization, reproduction, development and evolution of artificial life-like systems.
Virtual Community or Inhabited Virtual World finds its roots in the earliest text-based multi-user games: MUDs [3]. A detailed description of MUDs is given in [21] and a huge amount of well structured information and resources on actual MUDs can be found on the [1] website. Continuing the trend was the development of MOOs

---

[1] MaDViWorld stands for Massively Distributed Virtual World, see [9], [8], [13] and [10].
[2] The word Avatar comes from the Sanskrit: "Earthly incarnation of a Hindu god or goddess". The reference is in particular to a God called Visnu that was able to reincarnate himself through several and different faces. In the Internet the word Avatar is used to describe the "object" representing the user.
[3] Multi-User Dungeons

(adding object-oriented features to the MUDs), IRC[4] and conferencing systems, and the World Wide Web and its many progeny in the 1990s. Finally "inhabited"2D and 3D virtual spaces in Cyberspace have risen by the merging of text-based chat channels with a visual interface in which users are represented as "avatars". Virtual Community focuses on 3D worlds, 3D avatars and essentially on social interaction and chat. Examples of such approaches can be found in [5].

1.3   Our Approach

Now, that some different aspects of the Virtual Worlds field of research have been introduced, it is important to clarify the position that has been adopted by our research group for the MaDViWorld project. Our main concern is to develop a software platform supporting *distributed multi-user virtual worlds* populated with active objects. The definition given by Diehl in [4] suits us well:

> *Virtual worlds* are computer-based models of three-dimensional spaces and objects with restricted interaction. A user can move through a virtual world and interact with those objects in various ways.
>
> A *multi-user* virtual world is a virtual world where several users interact at the same time. These users work at different computers which are interconnected. In multi-user worlds, avatars play a central role.
>
> A virtual world is *distributed* if active parts of it are spread throughout different computers in a network. There is no need for a single host to have full knowledge of the world.

While developing distributed virtual worlds, one has to face both the problems of virtual reality and those of distributed systems.

According to the different concepts introduced in the first part, our approach is definitely closest to the Virtual Community, as we are clearly not concerned with the design of 3D immersive graphical spaces, neither with the simulation of living organisms. Contrarily to most Virtual Communities, in MaDViWorld the 3D aspects and the social interaction issues are not the first priority. Awareness and interaction between the users of the world is achieved and the virtual world paradigm is supported without sophisticated 3D features, which could, of course, be introduced later as graphical improvements. Indeed, the main originality of our approach is the customization of the virtual world, by populating it with active and mobile objects. Our goal is to enhance the concept of virtual world objects that appeared in MOOs, where the user could slightly adapt these objects thanks to basic scripting languages. The second goal of MaDViWorld is on the software architecture level. The technological challenge was to have a totally distributed, self-healing, self-configuring architecture supporting the virtual world paradigm. Such an architecture combines the great advantages (scalability, robustness,...) found in the document paradigm with the shared virtual world metaphor and contrasts with the "many clients-one server"architectures. As it seems that most of the existing virtual worlds platforms adopt a centralized, or partially centralized approach, it is interesting to tackle a completely distributed alternative. Some similar approaches are presented by [6] in URBI et ORBI, by [16] in MASSIVE or by [7] in DIVE.

---

[4]Internet Relay Chat

2   Conceptual Approach

The first part of this section is dedicated to a short typical scenario, which should be possible in a virtual world. Building on this story, we then identify and extract the main concepts that are involved and we present the MaDViWorld model.

2.1   The Vision

Suppose we have a virtual world, a user wants to "live"in. As the user strolls through the world, she discovers it and its components, objects and other users like her. Suddenly, she finds an interesting object, a fibonacci number calculator. She can use it to compute some numbers of the famous series. Then, she goes on with her world discovery. Doing so, she comes to a place where she sees two other users playing a "tic-tac-toe"game and a little crowd watching them. She joins the observers and, after a while, she says to her neighbour: "Do you want to play this game with me?" As the other agrees, they go to another place where the user has placed a copy of the previous board game and they start to play. When they are finished she puts the "tic-tac-toe" object back in her bag and leaves the other user.

2.2   The Key Concepts

The main concepts emerging from the above simple scenario are:

- **Avatar**: The human user needs a representation in the virtual world and is therefore personified by an avatar. Through her avatar the user can walk, "fly", look around the virtual world, manipulate objects and perform virtual actions. In other words, avatars allow users to interact with the virtual world and other users and also allow navigation through the world. We see one's avatar as being her representative in Cyberspace. In text-based virtual realities, such as MUDs, one's avatar consists of a short description which is displayed to other users who have their avatars "look" at her. In a 3D graphical world, the avatar can take the shape of an animated cartoon or of your favorite fantasy hero.

- **Object**: In the discussed virtual world, there are objects, not just simple passive data objects, but active objects the avatar can execute (e.g. the fibonacci calculator). These objects can even be multi-user (e.g. the two players "tic-tac-toe" board) and can have many observers like in our little story. Last, but not least, objects can be copied and/or moved by the avatar.

- **Location**: Avatars and objects have a location. This concept is natural and necessary, since it supports navigation.

- **Navigation**: It is the action of going from a given location to another. Avatars can perform this task, either if there is a **link** between these two locations, or directly if they know the **address** of the subspace.

- **Subspaces**: Each location can be seen as a subspace of the whole virtual world. It is natural to consider that the shared virtual space is composed of many different subspaces. Furthermore, the avatars and the objects are always **contained** within one given subspace.

- **Event**: The avatar has to be aware of its environment. This awareness is achieved by the concept of events. Another avatar entering one's subspace or a move in the "tic-tac-toe" game are simple examples of such events.

- **Event producer**: An event always has a source which produces it. For instance, the game could produce a "game finished event".

- **Event consumer**: An event can be catched and interpreted by an event consumer, which then reacts properly or simply ignores it. For instance, the players and the audience of a given game understand that the game is finished.

- **Event propagation**: Each event has an event propagation space. This space is a delimited zone around an event producer, within which an event consumer will be aware that the event has occurred.

The concepts above have been formalized and integrated within a general theoretical model (see [12]). The rather mathematical formalism used in the latter, however, is out of the scope of the present paper and not necessary for its further comprehension. Therefore, the next subsection provides a more practical and concrete description of the adopted model.

2.3   The **MaDViWorld** Model

Let us use the metaphor inherited from the MUDs for the subspaces: they are called *rooms* and the links between them are naturally called *doors*.
In order to avoid the dependance on a central server, the rooms have no "geographical" location relative to the entire world. Furthermore, within our actual prototype, the avatars and objects contained in a given room also have no location relative to the latter. In this basic model, when the avatar is in a given room, it only sees three lists containing respectively the available doors to other rooms, the present avatars and the present objects (cf. Figure 1). This is the simplest topological model that offers the feeling of being in a space with other users and objects, i.e immersion.
Figure 2 illustrates the conceptual model of a simple running world composed of four rooms and inhabited by three avatars. One can see that one of the rooms contains an active object.
It is also worth noting that, in order to maintain a consistent view of the world from its various components (i.e. avatars, rooms and active objects), a distributed event model is necessary. The basic mechanism is inspired by the observer pattern [15]. There are event producers and event consumers. The event consumers register event listeners to event producers, also called event sources, in order to be notified by them of events of interest. Figure 3 summarizes this mechanism.

3   A Concrete Implementation: The **MaDViWorld** Framework

This section shows why and how our concrete solution is implemented as an object-oriented framework. The first subsection discusses our technological choices. The second one gives a global view of the software architecture. The last three subsections focus respectively on the object structure, the distributed event model and the deployment of the different involved applications.
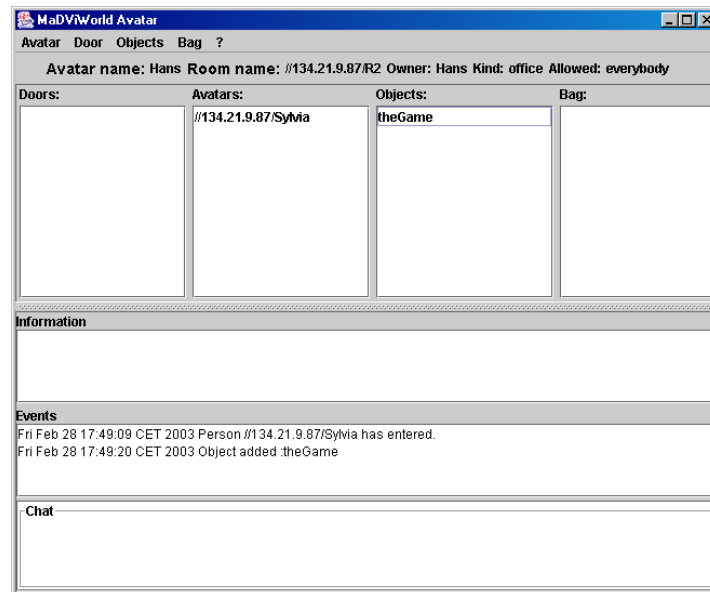
Figure 1: A MaDViWorld "basic" avatar user interface: *Its name is Hans and it is located in room R2. Avatar Sylvia is also present, as well as an active object called theGame. Hans has nothing in his bag. Sylvia just entered and deposited the active object from her bag.*
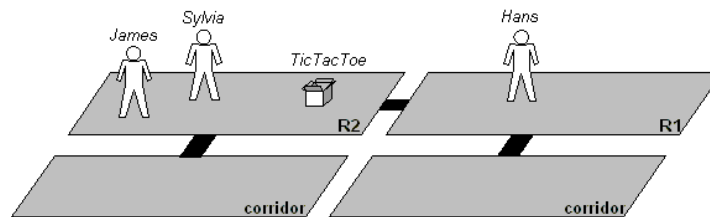

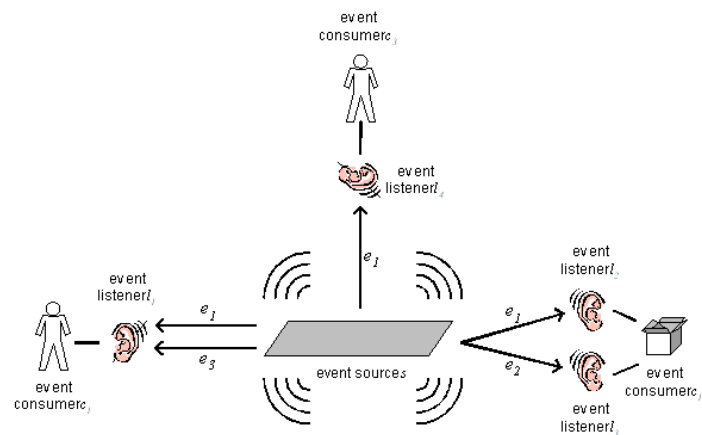Figure 2: The conceptual view of a simple world.

Figure 3: An event source and its listeners.

Avatar	
`getInformation();`	Returns the description of the avatar.
`getCurrentRoom();`	Returns the current room where the avatar is.
Object	
`getContainer();`	Returns the container of the object.
`setContainer();`	Sets the container of the object.
`getUI();`	Gets a graphical user interface to interact with the object.
Room	
`addObject();`	Sets a new object into the room.
`getObject();`	Returns a given object in the room.
`removeObject();`	Removes a given object from the room.
`getObjects();`	Returns a list of the objects in the current room.
`connect();`	Sets a given avatar into the room.
`disconnect();`	Removes a given avatar from this room.
`getAvatars();`	Returns a list of the avatars in the current room.
`addDoor();`	Adds a door to a given room.
`getDoors();`	Returns a list of doors to other rooms.
Event Producer	
`register();`	Registers an interested event consumer with this producer.
`unregister();`	Unregisters a registered event consumer with this producer.
Event Consumer	
`notify();`	Notifies the event consumer of an event.

Table 1: Some important method candidates of the main classes.

3.1 A Distributed Object-Oriented Framework

3.1.1 Why Object-Oriented?

From the key concepts identified in the previous section, several considerations can be done. First of all, since one of our main concern is to populate the world with an ever growing set of active objects, the object-oriented technology seems to be the natural way to face our implementation problems.

We further identify three major actors: the rooms, the avatars and the active objects. In order to keep the consistency of the world, two roles related to the events are associated to these three major actors: event producers and event consumers. At this stage, the services that these five components should provide can be roughly sketched. Object-oriented technology also fits well here, since one can define a set of interfaces or abstract classes, that can be implemented or specialized in a further stage through an inheritance "is-a" relation. This set of interfaces defines a common communication protocol between the different components of the world. It is briefly sketched by Table 1. For sake of simplicity, detailed methods signatures have been omitted. One can easily see that the three main abstract classes or interfaces defining the main actors of virtual worlds are certainly: `Avatar`, `Room` and `WObject`[5]

---

[5]`WObject` is used in order to avoid collision with the Java root class `Object`. The `W` stands for "world".

3.1.2 Why a Framework?

A software solution supporting our virtual world metaphor must take into consideration the two following issues:

- The extensibility of the conceptual model has to be supported. Several kinds of rooms can coexist in the same virtual world and different sorts of avatars should allow the users to discover these rooms. For example, one can imagine the integration of 2D and/or 3D rooms and of sophisticated avatar applications supporting local topological information. It might also be interesting to distinguish between public rooms and private rooms, containing sensitive objects and for which the security has to be reinforced.

- Furthermore, the customization of the world by populating the rooms with active objects is one of the main concerns of the project.

In order to satisfy these requirements of high adaptability, we adopted a layered software framework approach, leading from abstract to always more concrete classes. A *framework* is defined as a partially complete system that it is intended to be instantiated. It defines the architecture for a family of systems and provides the basic building blocks to create them. It also defines the places where adaptations for specific functionalities should be made. Deeper discussions about frameworks can be found in [3], [18] or [20].

3.1.3 Why Distributed?

In order to respect our initial goal, i.e. creating extensible virtual worlds, potentially as large as the whole Internet community itself, the choice of a well established and portable distributed technology was of the upmost importance.
In a massively distributed world, the subspaces are distributed on an arbitrarily large amount of machines. The only requirement is that each machine containing a part of the world runs a small server application and is connected to other machines through an appropriate network (e.g. the Internet). With Figure 4 it becomes clear that the simple virtual world of Figure 2 can be supported by many physical configurations. For instance, four machines interconnected by a network, each hosting one ore several applications (room server and/or avatar application). And the most relevant point is, that there is no central server.
The network distributed aspects and the fact that each part of the world should be able to run on different hardware platforms are the two main reasons for implementing our framework in Java. Other aspects like Java RMI dynamic classloading facilities (cf. [17]) and the interesting Jini technology (cf. [19]) also influenced our choice.
The MaDViWorld object-oriented software framework is presented in the next subsection. The different places where adaptations should be made, i.e. the *hot spots*, are identified and explained.

3.2 Global View

An overview of the whole framework is shown in Figure 5. First, let us consider an *horizontal decomposition*:

- The upper layer of the framework defines the communication protocol between the different components.
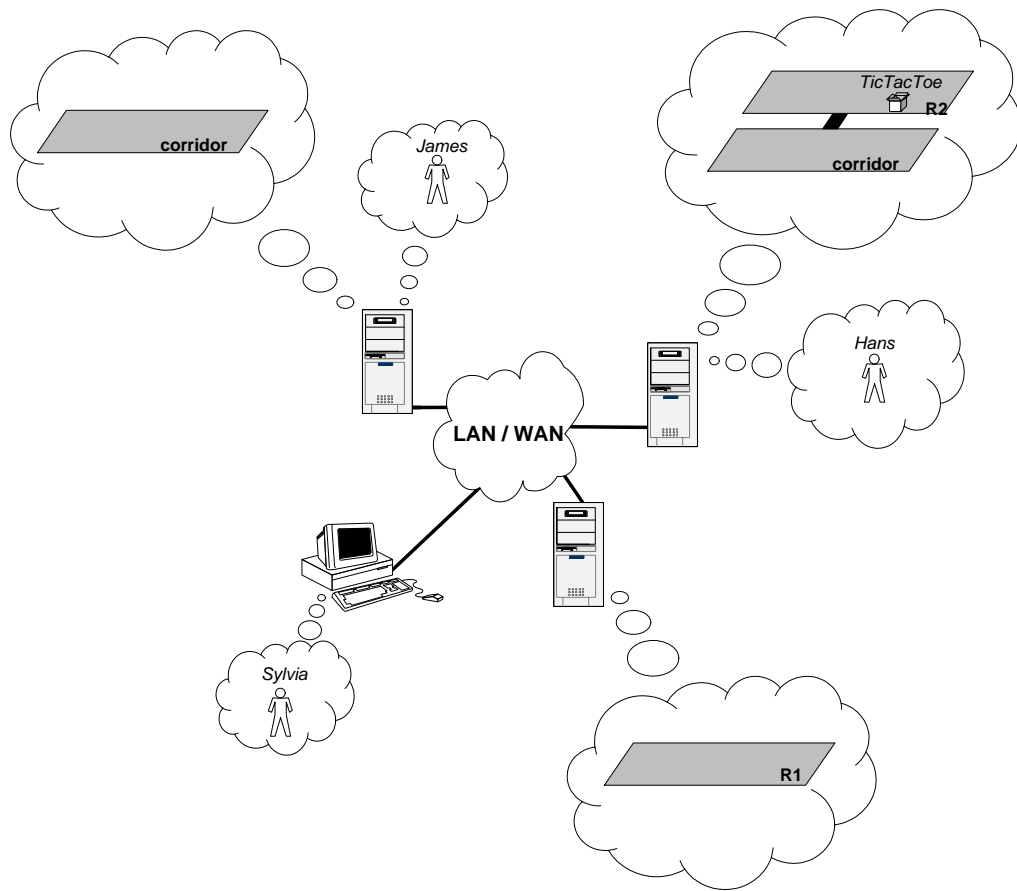
Figure 4: One possible physical configuration for a simple world.

- The middle layer consists of the default implementation packages of the framework.

- The lower layer, finally, is for the concrete applications, where all the application specific classes are placed.

Second, let us decompose the blocks of Figure 5 *vertically*. From left to right one finds respectively all the packages and classes relative to the *client* application (i.e. avatars), then those relative to the *server* application (i.e. rooms) and those relative to the active *objects* populating the rooms. Finally, there are two *utility packages*, the event package and rightmost one containing packages and classes used by the framework (such as http file servers, custom classloaders, etc.).
Obviously, the *Avatar*, *Room* and *Object* packages are the hot spots of the MaDViWorld framework. The framework user can:

- define and develop room server and/or avatar application by extending the framework classes in order to provide his own features.

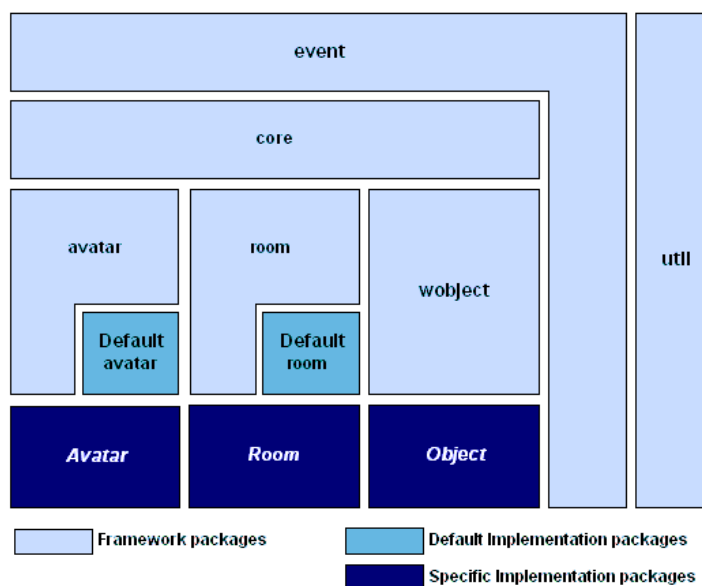- develop new types of objects by providing the appropriate implementations of the `wobject` package.

Figure 5: Overview of the MaDViWorld framework.

3.3   Active Objects Implementation

The purpose of this subsection is to better understand the process of objects creation and use. Figure 6 illustrates how the `wobject`[6] package has to be implemented in order to develop new objects.

---

[6]The root package for all MaDViWorld classes is `ch.unifr.diuf.madviworld`, but in this paper this is omitted for convenience reasons.

In order to add a new object, the framework user has to create the corresponding `wobject.newobj` package, which must contain two subpackages, one for the object's *implementation part* and one for its *graphical user interface* (GUI) *part*. This clean separation between the user interface and the object logic does not provide a two-way communication channel between these two parts. The client relationship between the `MyObjGUIImpl` class and the `MyObj` interface provides a one-way communication channel (from GUI to the implementation), but the implementation part cannot send information back to the GUI. The distributed event model designed to address this issue is presented below. This approach was chosen in order to allow an object implementation part to have arbitrarily many graphical user interfaces "observing" it.

For further comments about this package the reader is invited to consult the Object Programmer's Guide (cf. Appendix A). An insight of the possibilities of the MaD-ViWorld framework, the presentation of all the categories of existing objects, and a comparison between the objects of the virtual world and agents can be found in [11].
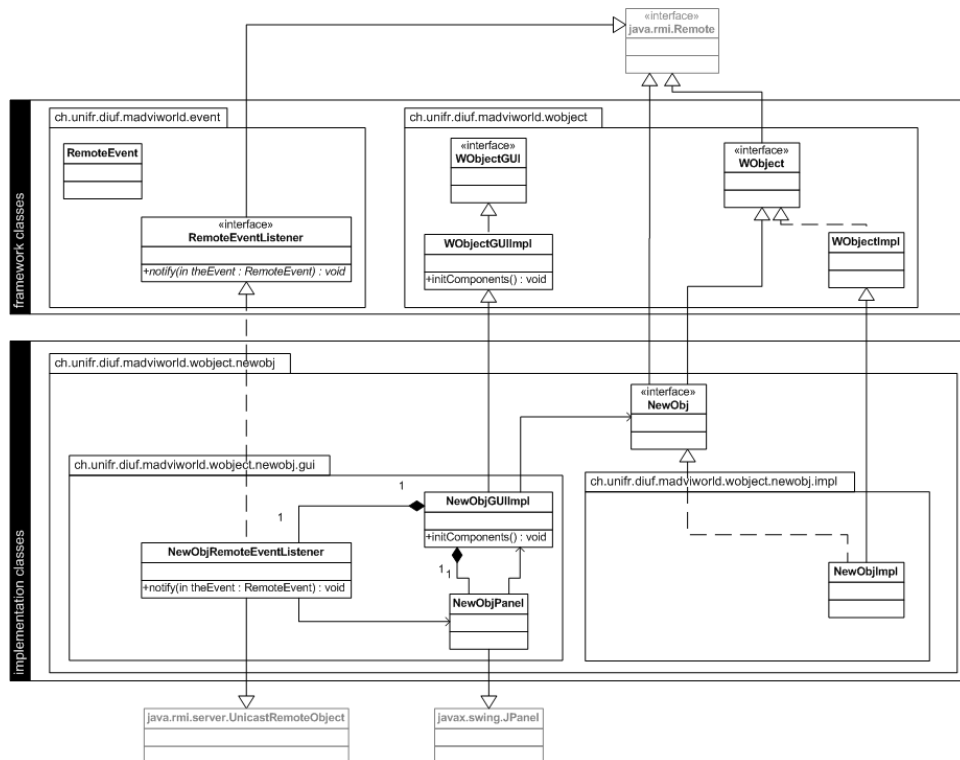

Figure 6: Implementation of the `wobject` package

3.4 The Distributed Event Model

Events play a crucial role in the MaDViWorld framework. Schematically, each time the state of one of the world components changes, a corresponding event is fired by the altering subject and consumed by the registered listeners, which react appropriately.

The management of all these events is a complex task:

- They are in reality remote events and several network problems can occur;

- Some of the events have to be fired to only a subset of all the listeners;

- Some listeners may not be interested in every type of event.

Thus the framework must offer a *distributed event model* that handles all these situations.

The two last points listed above, lead to the elaboration of an abstraction for creating unique identifiers. DUID is the acronym for Distributed Unique ID and is implemented in the `core.DUID` class[7]. Each room, object or avatar has an associated DUID that is generated by the framework, so that it can be identified without ambiguity. The use of such a DUID was inspired by [14].

It is now time to take a closer look at the content of the `event` package (see Figure 7) and how it solves the mentioned problems:

- The `RemoteEventListener` interface simply extends the `java.util.EventListener` interface and defines the single `notify()` method. Any object that wants to receive a notification of a remote event needs to implement it.

- The `RemoteEventProducer` interface defines the methods needed to register, unregister and notify event listeners used to communicate between different parts of the system. The register method takes as parameter the event type the listener is interested in. There are five possibilities: *all events*, *avatar* events, *room* events, *object* events and *"events for me"*. With the latter, the listener is only informed of events addressed explicitly to it (thanks to its DUID), without paying attention by whom.

- The `RemoteEvent` class defines remote events passed from an event generator to the event notifiers, which forward them to the interested remote event listeners. A remote event contains information about the kind of event that occurred, a reference to the object which fired the event and arbitrarily many attributes.

- The `RemoteEventProducerImpl` class implements the `RemoteEventProducer` interface.

- The `RemoteEventNotifier` helper class notifies in its own execution thread a given event listener on behalf of an `RemoteEventProducerImpl`.

Figure 7 shows the *design pattern* used through the whole framework for the collaboration between the three different parts of MaDViWorld (i.e. avatars, rooms and objects) and the utility `event` package. Note that the three of them are both implementing the `RemoteEventProducer` interface and are client of its implementation, `RemoteEventProducerImpl`. The operations defined by the interface are just forwarded to the utility class. With this pattern we have the suited inheritance relation (a `WObject` "is-a" `RemoteEventProducer`) without duplicating the common code. A lot of similarities with both the Proxy and the Observer Pattern can be found. For

---

[7]The DUID is the combination of a `java.rmi.server.UID` (an abstraction for creating identifiers that is unique with respect to the host on which it is generated) and an instance of `java.net.InetAddress` (a representation of the host's IP address where the object was created), which makes the DUID globally unique.

the latter, the push model is applied by the subjects, which send observers detailed information about their state change. Furthermore the subject informs only those observers that have registered interest in that event. These patterns are defined in [15], which also contains the preceding discussion.
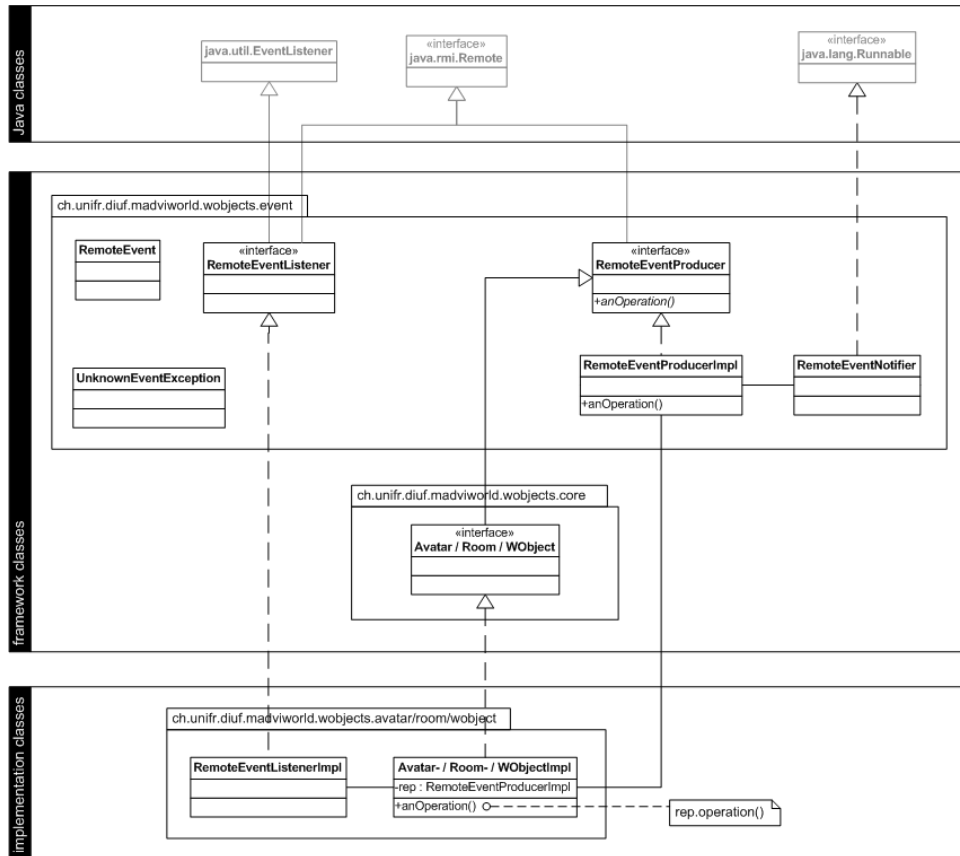

Figure 7: Pattern used for integrating the event model in the framework

To sum up the whole event mechanism, the UML sequence diagram of Figure 8 dwells on all the operations, from the registration phase to the firing and notification of an event. First (a), the event consumer registers a `RemoteEventListener` to a room, avatar or object whose events it is interested in. Second (b), due to a state change an event is fired and all interested listeners are notified, each by a `RemoteEventNotifier`. The informed listener can then do the appropriate work with regard to the type of the event. On Figure 8, one can also see the different methods invoked remotely across the LAN. This pattern present some similarities with the *Jini distributed event programming model*, which is specified in [2] and thoroughly explored in [19].
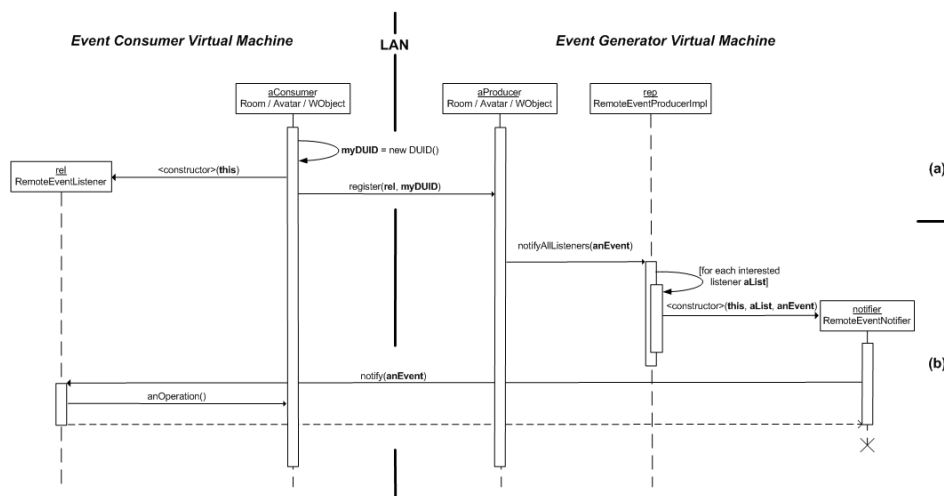
Figure 8: (a) Setup of the event model (b) Notification of an event

3.5 Deployment

There are three different kind of applications composing a MaDViWorld running environment: avatars, room servers and room setup utilities. They can be deployed independently on several machines.

The avatar application allows the end-user to log in the room of her choice and to explore the virtual world. It has to be deployed with the packages shown on Figure 9.
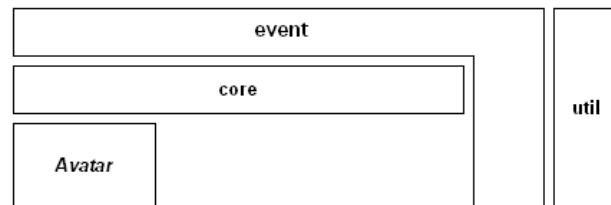

Figure 9: Packages needed for the deployment of avatars

The room server application, hosts different rooms containing objects. It ensures the persistence of the part of the world it is hosting. This application is deployed with the packages shown in Figure 10.

The third application, is the room setup utility. It allows for room creation and customization on a running server. The end user can create new rooms and add new objects to them by using it. It is deployed with the packages shown in Figure 11.

3.6 Further details

MaDViWorld is a collection of approximately 150 classes, organized in 8 main packages and several subpackages, representing more than 17'000 lines of Java source code.
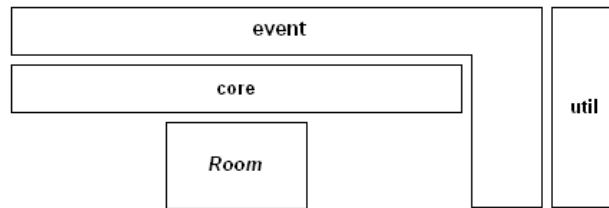
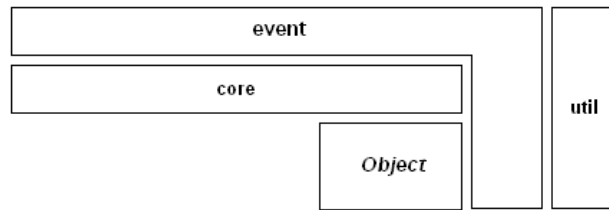Figure 10: Packages needed for the deployment of room servers

Figure 11: Packages needed for the deployment of the room setup utility

Therefore, a deeper insight of its structure is out of the scope of this paper. But the interested reader is referred to [10] for details about the packages responsible for the event handling, about the object structure or the lookup mechanisms. Further details, concerning technical aspects like the object mobility or the state recovery mechanism for the persistence of the world, that were already present in the first version of the framework are discussed in [8]. Up-to-date evolutions of the project are published on its official website (cf. Appendix A).

4 Conclusion

Our conclusion is divided into two subsections. While the first briefly summarizes the MaDViWorld framework main benefits, the second concentrates on the work and research yet to be done.

4.1 Achievements

The actual version of MaDViWorld is a fully functional framework for creating highly distributed virtual worlds. It offers programmers the opportunity to transparently develop any kind of objects and test them in a virtual world, with all the advantages of mobility, remote execution or persistence. It already proved its strong points through extensive use in various students projects. This experiences helped improving the design and adding features, that were missing in the very first version of the prototype. Some of the most interesting ones are:

- a highly parameterable *distributed event mechanism* is provided by the framework and is the cornerstone for all interactions between avatars, objects and rooms.

- in order to locate objects and services (room servers, rooms, avatars) the framework provides a *dual lookup mechanism*. On the one hand, it makes use of rmiregistries provided with Java RMI. On the other hand, it also allows rooms and avatar to register themselves to remote Jini lookup services (reggie) and to act as well-behaved Jini services. This feature offers great benefits for finding rooms and avatars when one has no *a priori* knowledge of the already running world (i.e. on which machine to find a room for example). In addition to that, the template matching mechanism of Jini lookup presents a lot of possibilities (e.g. searching all german speaking avatars ar all rooms intended for games).

- The transport of *mobile objects* from one room to another is complex because it occurs in fact often from one machine to another. All the nitty gritty is kept transparent for the end-user and hidden from the framework user.

- Avatars can execute an object remotely, but the graphical user interface of the object always runs locally. As the avatars has no *a priori* knowledge of the kind of objects they will meet during their trip through the world, the framework provides a *graphical user interface transport mechanism*.

4.2   Future Work

The MaDViWorld framework is a solid starting point for several future improvements and it serves as testbed for several research projects that have already started and which cover several directions:

- On the abstract and theoretical side, a *security model* has to be developed that will complete the model presented in Section 2. Thus it will be possible to develop a mechanism that defines the rights each avatar has to interact with the world. On the practical side, this security model should be integrated in the existing framework.

- Objects, as they are complete Java programs, offer a great degree of customization to the framework user, who will then "inject" the object into the virtual world. But the end user should have more than parametrization to change the object's behavior at runtime. Hence, if she has basic programming skills she could dynamically adapt some objects offering *scripting facilities*.

- Rooms with more complex *2D or 3D aspects* should be provided.

- It would also be interesting to have *multimedia objects* (sound and/or video streaming) to populate the rooms.

- *Static XML based descriptions* of the rooms and their attributes, as well as the connections between them should be developed. For avatar and the information describing them this would also be useful. One of the goals of such developments is to enhance the world creation, which could be achieved "on the fly" by a parser tool. Persistence and state recovery mechanisms could then take advantage of this feature.

- Concrete practical applications of entire virtual worlds supported by the MaDViWorld framework will be proposed. Developing educational worlds as virtual campus could be a possible *"killer application"*. Such a world is hinted in [9].

All these enhancements, should bring the MaDViWorld framework to an even greater level of maturity.

References

[1] The mud connector `http://www.mudconnector.com`. [on line], accessed on 7th May 2004.

[2] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification.* The Jini Technology Series. Addison-Wesley, 1st edition, 1999.

[3] F. Buschmann, R. Meunier, and H. Rohnert. *Pattern-Oriented Software Architecture - A System of Patterns.* John Wiley and Sons, 1996.

[4] S. Diehl. *Distributed Virtual Worlds: Foundations and Implementation techniques using VRML, Java and CORBA.* Springer, New York, 2001.

[5] J.-C. H. (ed.). *Virtual Worlds: Synthetic Universes, Digital Life, and Complexity.* Perseus Books, Reading, Massachusetts, USA, 1999.

[6] Y. Fabre, G. Pitel, L. Soubrevilla, E. Marchand, T. Graud, and A. Demaille. A framework to dynamically manage distributed virtual environments. In J.-C. Heudin, editor, *Virtual Worlds*, volume 1834 of *Lecture Notes in Computer Science*, pages 54–64. Second International Conference, VW 2000, Paris, France, July 2000, Springer-Verlag, 2000.

[7] E. Frécon and M. Stenius. Dive: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments*, 5(3):91–100, June 1998.

[8] P. Fuhrer, G. K. Mostéfaoui, and J. Pasquier-Rocha. The MaDViWorld software framework for massively distributed virtual worlds: Concepts, examples and implementation solutions. *Department of Informatics Internal Working Paper no 01-23, University of Fribourg, Switzerland*, 2001.

[9] P. Fuhrer, G. K. Mostéfaoui, and J. Pasquier-Rocha. MaDViWorld: a software framework for massively distributed virtual worlds. *Software - Practice And Experience*, 32(7):645–668, June 2002.

[10] P. Fuhrer and J. Pasquier-Rocha. Massively distributed virtual worlds: a framework approach. *Department of Informatics Internal Working Paper no 02-16, University of Fribourg, Switzerland*, 2002.

[11] P. Fuhrer and J. Pasquier-Rocha. Madviworld objects: Examples and classification. *Department of Informatics Internal Working Paper no 03-15, University of Fribourg, Switzerland*, 2003.

[12] P. Fuhrer and J. Pasquier-Rocha. Massively distributed virtual worlds: A formal approach. *Department of Informatics Internal Working Paper no 03-14, University of Fribourg, Switzerland*, 2003.

[13] P. Fuhrer and J. Pasquier-Rocha. Massively distributed virtual worlds: A framework approach. In E. A. Nicolas Guelfi and G. Reggio, editors, *Scientific Engineering for Distributed Java Applications*, volume 2604 of *Lecture Notes in Computer Science*, pages 111–121. International Workshop, FIDJI 2002 Luxembourg-Kirchberg, Luxembourg, November 2002, Springer-Verlag, March 2003.

[14] A. Gachet. A software framework for developing distributed cooperative decision support systems - construction phase. *Department of Informatics Internal Working Paper no 02-02, University of Fribourg, Switzerland*, 2002.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, Massachusetts, 1995.

[16] C. Greenhalgh and S. Benford. MASSIVE: A distributed virtual reality system incorporating spatial trading. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95)*, pages 27–35, Los Alamitos, CA, USA, 30– 1995. IEEE Computer Society Press.

[17] W. Grosso. *Java RMI*. O'Reilly & Associates, Inc., 2002. Designing and building distributed applications.

[18] C. Larman. *UML and Patterns*. Prentice-Hall PTR, 2002.

[19] S. Li. *Professional Jini*. Wrox Press Ltd., 2000.

[20] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, 1995.

[21] E. Reid. Cultural formations in text-based virtual realities. Master's thesis, University of Melbourne, January 1994.

A The Official Website

The official website of the MaDViWorld project is:
`http://diuf.unifr.ch/softeng/projects/madviworld/`

This site provides the following information:

- the whole source code of the latest MaDViWorld version, which can be freely downloaded;

- an installation guide;

- an online Object Programmer's Guide;

- the up-to-date javadoc documentation of MaDViWorld ;

- actual publications related to MaDViWorld in electronic format;

- a list of completed and ongoing related student projects.