MaDViWorld : a Software Framework

for Applying a Collaborative Virtual World Paradigm to the Internet

Patrik FUHRER
Department of Informatics, University of Fribourg
1700 Fribourg, Switzerland
patrik.fuhrer@unifr.ch

and

Jacques PASQUIER-ROCHA
Department of Informatics, University of Fribourg
1700 Fribourg, Switzerland
jacques.pasquier@unifr.ch

ABSTRACT

MaDViWorld is an object oriented software framework supporting the implementation of fully distributed virtual worlds on the Internet. While the World Wide Web proposes a document paradigm with HTTP servers containing documents consulted by users with the help of browser applications, MaDViWorld supports a much richer paradigm based on room servers hosting spaces populated by full-fledged objects, that avatar applications can activate, move and share transparently. Nevertheless, the dissemination of virtual worlds on the Internet suffers from two main weaknesses: (1) they are typically based on centralized architectures and do not scale well; and (2) they usually propose a rather closed software environment with limited extension and programming facilities.

Within this context, the MaDViWorld project main goal is to provide its users with the appropriate software environment for creating all kinds of new collaborative objects and for sharing them transparently with others. The present paper illustrates this process with several examples from projects recently accomplished at the DIUF (Department of Informatics of the University of Fribourg, Switzerland) and shows how MaDViWorld provides the hooks for taking care of some of the most challenging distributed virtual world problems such as managing event propagation and securing access to ressources.

Keywords: Virtual World, Collaborative Work, Mobile Objects and Distributed Software Framework

1. INTRODUCTION

The Software Engineering Group at the DIUF has developed an object oriented distributed framework supporting massively distributed virtual worlds, called MaDViWorld.

The goal of this paper is to illustrate how MaDViWorld technology can be used on the Internet in order to apply a much richer collaborative paradigm than the classical document one proposed by the World Wide Web (WWW). The paper is organized as follows. Section 2 reviews some of the main virtual world concepts such as subspaces, avatars and objects and further explains the paradigm shift between classical WWW browsing and MaDViWorld usage. Section 3 is more technical and presents a discussion of the adopted software architecture. Section 4 provides a sample of the objects that can be created within MaDViWorld. Finally, Section 5 summarizes the main achievements of this work and wraps up the paper by describing future possible collaborative worlds based on MaDViWorld technology.

2. BASIC VIRTUAL WORLD CONCEPTS

For the further comprehension of this paper, the following four terms need to be briefly explained:

1. *Avatars* are the virtual representation of the users. Concretely, an avatar is a tool that allows a given user to move through the world, to interact with its inhabitants and objects and that lets the other users know where she is and what she is doing. Among people working on virtual reality and cyberspace interfaces (see [7, 30, 31]), the word Avatar is used to describe the "object" (icon, two or three-dimensional photo, design, picture or animation) representing the user in a shared virtual reality. In other words, an avatar is an instantiation of the user's body in the computerized medium. In text-based virtual realities, such as

MUDs[1] and MOOs[2] (see [4, 28]), avatars consist of a short description which is displayed to the users whose avatars "look" at them.

2. In order to distinguish between near and distant elements, it is essential to divide the world into subspaces where the users might or might not enter and in which all interactions take place. Let us call such subspaces *rooms.*

3. Rooms are connected by *doors*, which an avatar can use for moving from one room to another.

4. *Objects* populate the rooms. They are not just simple passive data objects, but full-fledged objects (single or multi user) avatars can execute and share (e.g. games, whiteboards). Furthermore, in a distributed world, objects should be "physically" mobile, i.e. transparently movable from one room on a given server to another room hosted on a different machine. In MaDViWorld, mobility is either performed autonomously by the object itself or passively with the help of the avatar transporting it in her *bag*.

The conceptual model, that emerges from these considerations is illustrated below with the help of a simple typical scenario.

A Typical Scenario

The starting point is a virtual world composed of two rooms, R1 and R2, hosted on two different machines. Let us comment, step by step, the scenario illustrated by Figure 1.

- Figure 1a): The virtual world is shared by three avatars: James, Sylvia and Hans, all present in the same room R1. There is a battleship game object in this room.
- Figure 1b): Sylvia and Hans both launch the battleship game and start playing it.
- Figure 1c): James also launches the battleship game. As it is a two players game, he becomes an observer of the game and can only watch how his two roommates play.
- Figure 1d): Sylvia and Hans decide to finish their game in room R2. Sylvia takes the battleship object and puts it in her bag.
- Figure 1e): Sylvia and Hans move to the empty room R2. Sylvia puts the game she had in her bag into the room. Then both Hans and Sylvia reactivate the game and go on from the point they stopped before. James is now alone in room R1.
- Figure 1f): The game is finished and Sylvia logged off the world. James and Hans are still inhabiting the world, each in a different room.

Although very simple, the preceding story reveals several interesting points:

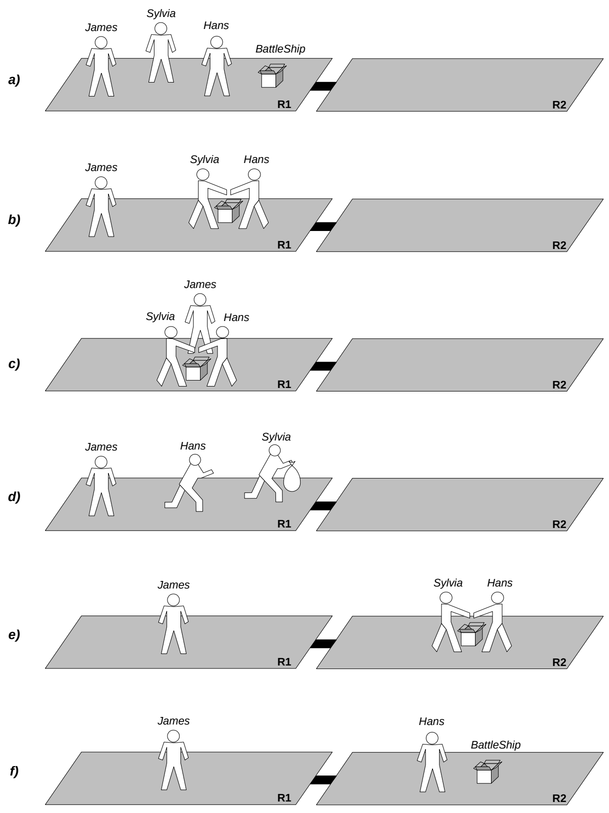


Fig. 1: A typical scenario in MaDViWorld

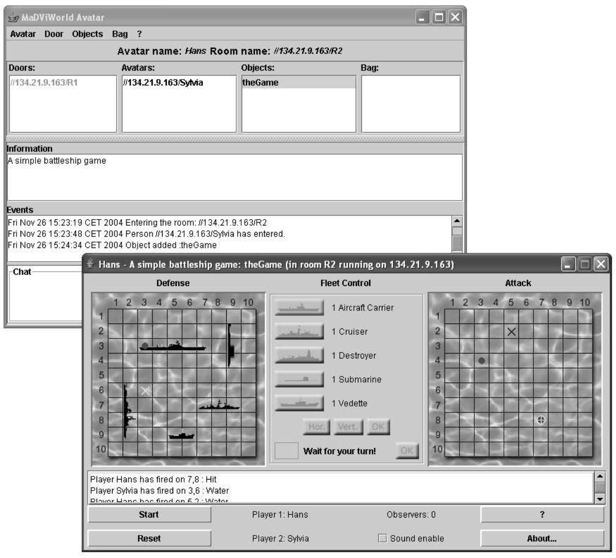
- MaDViWorld's powerful remote event mechanism plays an important role at two levels in this scenario. On the one hand, thanks to it, the avatars are aware of their environment. James immediately knows that Sylvia and Hans left the room. Hans sees when Sylvia puts the battleship object in room R2 (see Figure 2). On the other hand, the event mechanism is used to update the graphical user interface of the objects. This allows each move to be displayed immediately on each logged avatar's board, player or observer.
- The battleship object has "physically" been carried from room R1 to room R2 by the avatar Sylvia. Note that R2 is hosted by another machine than R1 and that the machine hosting R2 had no prior knowledge of this kind of object.
- The state of the game has not been lost during its transfer from R1 to R2.

Document versus Virtual World Paradigm

At this stage, it is worth comparing the virtual world paradigm just presented above with the document one usually applied when browsing the web:

- Within the *document paradigm*, documents, often active ones able to react to various user actions, are made available on one or several servers, and client applications (e.g., web browsers) can be used to interact with them. Typically, each user copies the documents onto

[1]MUD stands for Multi User Dungeon.
[2]MOO is the acronym for MUD Object-Oriented.



Avatar name: *Hans* Room name: *//134.21.9.163/R2*

Doors: //134.21.9.163/R1
Avatars: //134.21.9.163/Sylvia
Objects: theGame

Information: A simple battleship game

```
Fri Nov 26 15:23:19 CET 2004 Entering the room: //134.21.9.163/R2
Fri Nov 26 15:23:48 CET 2004 Person //134.21.9.163/Sylvia has entered.
Fri Nov 26 15:24:34 CET 2004 Object added :theGame
```

Hans - A simple battleship game: theGame (in room R2 running on 134.21.9.163)

```
Player Hans has fired on 7,8 : Hit
Player Sylvia has fired on 3,6 : Water
```

Player 1: Hans Observers: 0
Player 2: Sylvia

Fig. 2: Hans' avatar application and battleship object GUI

her local machine and her interactions with them have no direct repercussions on the other connected users. In particular, a user never directly modifies the original document. The underlying metaphor is the one of a huge cross-referenced book where each user browses through the pages totally unaware of other users performing the same task at the same moment. All actions are asynchronous and, thus, there is no need for a central server to coordinate user interactions with the pages of the book or to take care of an event redistribution mechanism. The main advantage of this approach is that it allows a truly distributed architecture with thousands of http servers interconnected all over the world. If a crash occurs, only the pages hosted by the failed or the no longer reachable servers become momentarily unavailable. The whole system is extremely robust and, since the connection of new decentralized servers is always possible, there is no limit to its growth.

- Within the *virtual world paradigm*, multiple users and active objects interact in the same space and therefore have a direct impact on each other. Within such systems, if a user interacts with an object, the other connected users can see her and start a dialog with her. Moreover, it is possible for a user to modify some properties of the world and all the other users present in the same subspace must immediately be made aware of it. It is worth noting that applications of the virtual world paradigm range from simple textual chat to sophisticated 3D virtual worlds (e.g. [26]) used for military simulations.

3. IMPLEMENTATION

At the implementation level, systems based on a distributed[3] virtual world metaphor are clearly the most complex ones. Indeed, the users interact directly with the original objects of the system and the resulting events must be correctly synchronized and forwarded in order to maintain the consistency of the world (see [24, 19]). To face these issues (i.e. supporting scalability when the virtual world grows very large and allowing for code mobility when objects are moving) virtual world developers have to choose carefully an appropriate software architecture. This section presents why and how MaDViWorld is implemented as an object-oriented framework. The first subsection discusses our technological choices. The second subsection gives a global view of the software architecture, while the three last ones concentrate on more specific topics: the programming of new objects, the distributed event model and security.

The interested reader is referred to [11, 13, 14] for a more detailed presentation of the MaDViWorld software architecture and of its theoretical foundation and to [17, 10, 8, 34] for a discussion of related systems.

A Distributed Object-Oriented Framework

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 distributed event model 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 method signatures have been omitted. One can easily see that the three main abstract classes or interfaces defining the main actors of virtual worlds are: Avatar, Room and WObject. The two interfaces, EventProducer and EventConsumer, define roles associated with these actors and will be further explained in the Distributed Event Model subsection.

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

[3]In the context of virtual worlds, "distributed" means that the architecture must not be limited to a single central server containing the whole virtual world and guaranteeing its consistency with many clients connected to it. It is imperative that distinct clusters of subspaces might be distributed on separate servers for scalability purpose.

Avatar	
getCurrentRoom();	Returns the current room where the avatar is.
WObject	
getContainer();	Returns the container of the object (e.g., a room).
setContainer();	Sets the container of the object (e.g., a room).
Room	
addAvatar();	Sets a given avatar into the room.
removeAvatar();	Removes a given avatar from this room.
getAvatars();	Returns a list of the avatars in the current room.
addObject();	Sets a new object into the room.
getObject();	Returns a given object in the room.
getObjects();	Returns a list of the objects in the current room.
removeObject();	Removes a given object from the room.
addDoor();	Adds a door to a given room.
getDoor();	Returns a given door in the room.
getDoors();	Returns a list of doors to other rooms.
removeDoor();	Removes a door from a given room.
EventProducer	
register();	Registers an interested event consumer.
unregister();	Unregisters a registered event consumer.
EventConsumer	
notify();	Notifies the event consumer of an event.
Event	
getSource();	Returns the source of the event.
getSeqNum();	Returns the sequence number of this event.
addAttribute();	Attaches a given attribute to the event.
getAttribute();	Returns a given attribute of the event.
getID();	Returns the ID of the event or the event type.

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

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. Let us briefly recall that a *framework*[4] is a partially complete system that is intended to be instantiated. It defines *(i)* the architecture for a family of systems and provides the basic building blocks to create them, and *(ii)* the places where adaptations for specific functionalities should be made.

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 utmost 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. 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 (see [18]) and the interesting Jini technology (see [21]) also influenced our choice.

[4]Deeper discussions about frameworks can be found in [5], [20] and [27].

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.

Global View

MaDViWorld is a *distributed* framework and adopts a multi-layered and multi-tiered architecture. More precisely there are abstraction layers and orthogonal deployment tiers. This decomposition allows for an optimal separation of concerns between the different building blocks. Figure 3 illustrates the global structure of the framework.

First, let us recall the roles of each abstraction layer, which altogether embody the fundamental principle called *separation of interface and implementation* [6]:

- The *upper abstraction layer* (core) contains the interface parts of all the main components of the system. It defines the functionality of each component and provides clients with guidelines for using them. The specification of these interfaces could be strengthened by using *Design by Contract* [23]. As MaDViWorld is implemented in the Java language which does not directly support *Design by Contract*, rigorous specification must be provided by a good documentation of the interface methods. Thus, this first layer defines clear boundaries between the components and defines a communication protocol between them.
- The *middle layer* consists of the default implementation packages of the framework. It contains the implementation part of the components and the actual code for the functionality they provide.
- The *lower layer* is for the concrete applications, where all the application specific classes are placed. This layer may provide specializations of the features provided by the middle layer.

The main idea behind this decomposition could be summarized with the following idiom: "Program against interfaces, not classes." Adopting this technique is a way to achieve information hiding and encapsulation and results in a low coupling of components. This approach supports changeability and eases the task of altering a component's behavior or representation. The Bridge [16] pattern, for example, addresses this principle.

Second, let us give some details about the vertical tiers which correspond to the three main applications interacting when using virtual worlds.

- *Avatar* application: This leftmost tier contains the classes and packages implementing the avatar. It is a client application allowing for the connection to rooms, and for the interaction with objects and other avatars. They basically play the same role as classical browsers within the World Wide Web and they do not necessarily need to present a sophisticated 3D interface.

- *Room Server* and *Rooms*: The second tier is composed of two parts. The implementation of the room interface supports a single room. The second component of this layer is dedicated to a room server application. Room server applications are set up on networked machines the same way as HTTP servers are for the web. A room server application allows for creating an arbitrary number of rooms with various properties on a given machine and for connecting them with others (including on different room servers) through doors.
- *Setup Application* and *Objects*: This tier contains the packages concerning the objects. A room setup application is a user-friendly wizard which allows for the creation and customization of rooms on distant room servers, and for the installation of objects into them.

There remain two building blocks that were not discussed yet: event and util. These are in fact two utility packages. The first one is dedicated to the remote event mechanism and the second one contains packages and classes used by all the components of the framework (such as http file servers, custom classloaders, etc.).

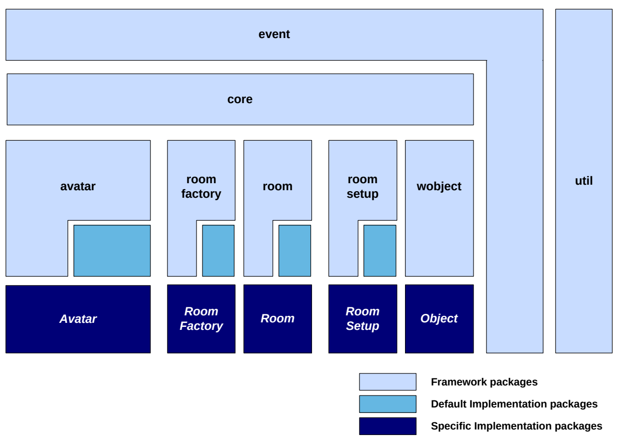


Fig. 3: Vertical and horizontal layers of the MaDViWorld framework

Each of the three main tiers can be deployed separately. The applications are deployed with the packages directly concerning themselves, as well as those common to all applications, i.e. the core layer, as well as the event and util packages.

Programming Objects

Objects occupy a special place in the distributed virtual world. At the user level, they aim to resemble as much as possible objects of the real world in terms of mobility. At the programmer level, objects are the main hot spot of the framework, since adding a new type of object is the most obvious way to customize an existing virtual world. This subsection explains the extension mechanism and the software design of the object related classes.

Objects must offer a graphical user interface (GUI) to the avatar who wants to use them. As the avatar and the object generally run on different computers, the GUI of the object must be executed on the avatar's host and remotely interact with the application logic of the object. To achieve this, a design pattern fostering a clean separation between presentation and logic is adopted.

Thus, when a developer wants to add a new object NewObj to the framework she has to separately provide[5] the three following pieces of code:

1. the classes supporting the *logic* of the object (see Figure 4). This is done by implementing a class (NewObjImpl), which extends the abstract WObjectImpl framework class;
2. the classes dedicated to the *presentation*, by extending WObjectGUIImpl (see Figure 5). This graphical class essentially serves as a graphical container of the JPanel subclass NewObjPanel. Hence the latter can directly be designed with any Integrated Development Environment (IDE).
3. the object's pure functionality, expressed via the methods of its NewObj interface. This interface is the coupling point between UI code and functionality code.

One advantage of this architecture, in which UI and functionality are loosely coupled, is that multiple UIs can be associated with the same object. Associating multiple UIs with one object lets you tailor different UIs for clients that have particular UI capabilities, such as Swing or speech. Clients can then choose the UI that best fits their user interface capabilities. In addition, you may want to associate different UIs that serve different purposes, such as a main UI or an administration UI, with an object.

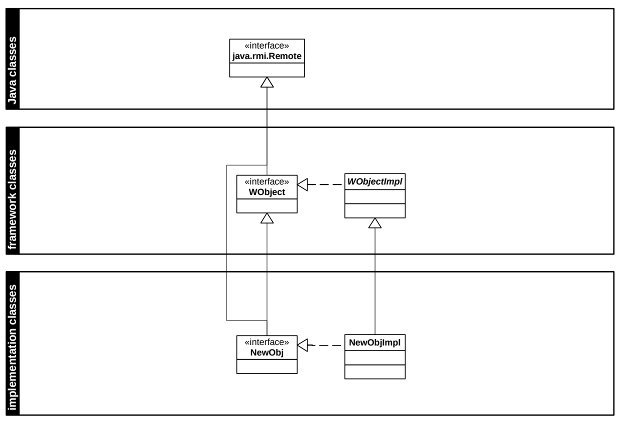


Fig. 4: Implementation of the logic part of an object

However this clean separation does not provide a two-way communication channel between these two parts. The aggregation relationship between the NewObjPanel class and

[5]For detailed instructions about how to create a new type of object the reader is invited to consult the MaDViWorld Object Programmer's Guide on the project's web site [12].

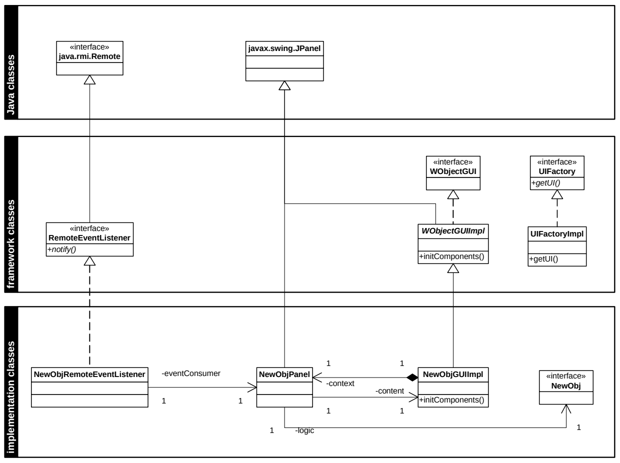


Fig. 5: Implementation of the presentation part of an object

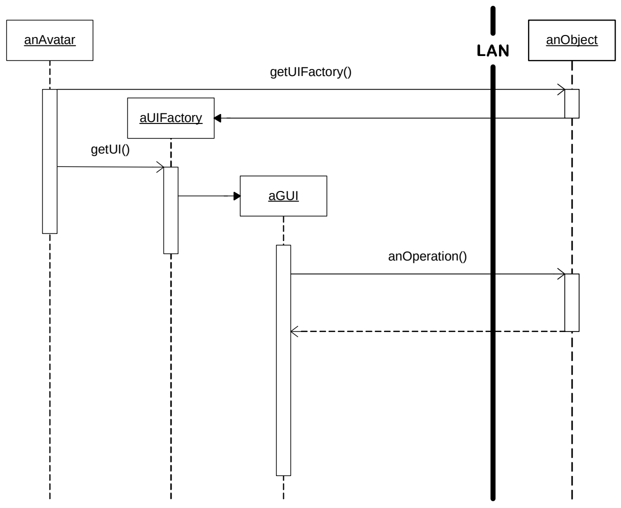


Fig. 6: An avatar getting a GUI to an object

the NewObj class provides a one-way communication channel (from the UI to the logic), but the logic cannot send information back to the UI. The distributed event model presented in the next subsection fills this gap.

Indeed, the UI registers the NewObjRemoteEventListener depicted on Figure 5 to the logic part of the object, which extends RemoteEventProducer (see Figure 7). This allows the object logic to easily notify the remote event listeners of the object's presentations. In this way, an object's logic part does not have to care about the presentation's implementation details. Furthermore, an arbitrary number of UIs can be attached to a single logic simultaneously. Thus, one has a solution which allows a given object to be shared by several avatars using it at the same time.

The sequence diagram of Figure 6 dwells on the mechanism that allows the avatar to get a GUI to a remote object, thus elucidating the role of the UIFactory[6]. This mechanism was inspired by one of the first successes of the Jini.org Jini Community Process, the ServiceUI project [32, 33], led by Bill Venners of Artima Software. The ServiceUI API enables multiple user interfaces to be associated with a single Jini service, allowing the service to be accessed by users with varying preferences and accessibility requirements on computers and devices with varying user interface capabilities.

The Distributed Event Model

Events play a crucial role in the MaDViWorld framework because they glue its different components together. Indeed, events are the only communication channel between rooms and avatars, rooms and objects and between two objects. Moreover the previous subsection showed yet another situation where remote events play a central role, namely, offering a communication channel from object logic to its UIs. Schematically, each time the state of one of the world components changes, a corresponding event is triggered by the altering subject and consumed by the registered listeners, which react appropriately. The management of all these events is a complex task for several reasons: *(i)* they are in reality remote events and several network related problems can occur; *(ii)* some of the events have to be fired to only a subset of all the listeners; *(iii)* some listeners may not be interested in every type of event. The *distributed event model* of the framework must handle 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 DUID class[7]. Each room, room server, object or avatar has an associated DUID that is generated by the framework and that never changes during its life cycle, so that it can be identified without ambiguity. The use of such a DUID was inspired by [15].

It is now time to take a closer look at the framework classes which aim to solve the mentioned problems (see Figure 7):

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

- The RemoteEventProducerImpl class implements two interfaces: *(i)* RemoteEventProducerRemote is an interface defining the methods that interested event consumers can remotely invoke to register their listeners; *(ii)* RemoteEventProducerLocal does not extend java.rmi.Remote since the methods it defines are not offered to remote clients. Therefore RemoteEventProducerImpl provides the methods needed to register, unregister and notify event listeners used to communicate between different parts of the system. The register method takes as pa-

[6]To allow the UIFactory to return a concrete GUI, some resources (e.g., sound files, icons, etc.) may need to be downloaded. For sake of simplicity, Figure 6 does not show how these resources are transferred.

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

rameter the event type the listener is interested in. There are five possibilities: *all* events, *avatar* events, *object* events, *room* 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 RemoteEventNotifier helper class notifies in its own execution thread a given event listener on behalf of a RemoteEventProducerImpl.
- The RemoteEvent class defines remote events passed from an event producer 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 design pattern illustrated by Figure 7 is 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 RemoteEventProducerRemote interface and are client of its default 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 the Proxy pattern defined in [16] can be found. This composition based design is more flexible and better adapted to our class hierarchy than the straightforward approach consisting of just inheriting of a common RemoteEventProducerRemote implementation. Anyway, the main inspiration of this structure comes from the Observer [16] pattern and its *publish-subscribe* interaction kind.

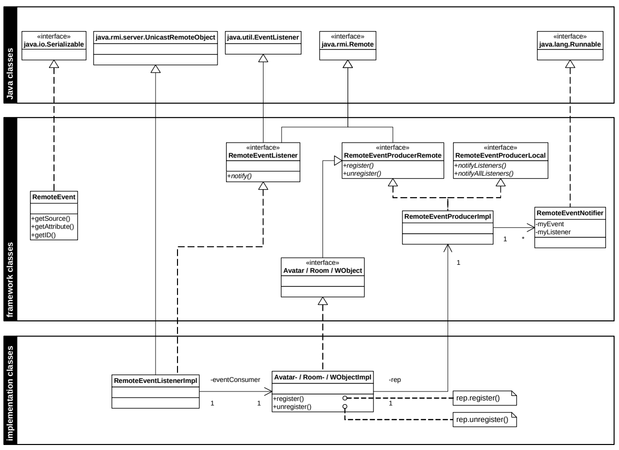


Fig. 7: Pattern used for integrating the event model in the framework

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 presents some similarities with the *Jini distributed event programming model*, which is specified in [2] and thoroughly explored in [21].

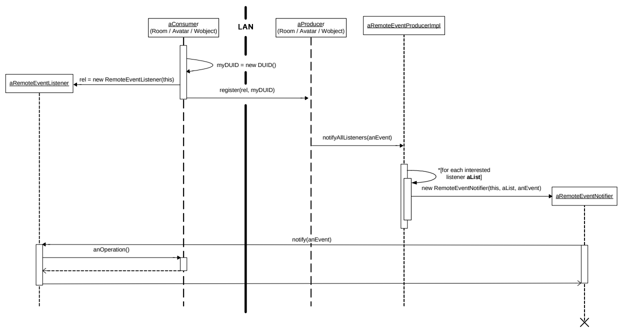

aConsumer (Room / Avatar / Wobject)
LAN
aProducer (Room / Avatar / Wobject)
aRemoteEventProducerImpl
myDUID = new DUID()
aRemoteEventListener
rel = new RemoteEventListener(this)
register(rel, myDUID)
notifyAllListeners(anEvent)
*[for each interested listener aList]
new RemoteEventNotifier(this, aList, anEvent)
aRemoteEventNotifier
notify(anEvent)
anOperation()

Fig. 8: (a) Setup of the event model (b) Notification of an event

Security

Security, privacy and trust are crucial elements in virtual world systems. One has to distinguish between two levels of security concerns: *(i)* the system level and *(ii)* the virtual world level. In order to address system level security concerns (e.g., passing through firewalls, encrypted communication protocol, downloaded proxy code trust, etc.), facilities offered by the Java and Jini technology can be used. In the actual version of the MaDViWorld project, system level security is not the first priority, and some further configuration would be necessary prior to large scale deployment. This section clarifies how the framework manages security at the virtual world level, i.e. security sensitive *actions inside the virtual world.*

There are several critical actions that objects and avatars may undertake while visiting the rooms of a virtual world: access a given room, use an object, remove or copy an object from a given room, etc. All these interactions concern a room and another entity (an avatar or an object).

Thus the basic principle of MaDViWorld 's security model is that *the subspace grants access rights or privileges to avatars and objects.* Rooms achieve this task by using *challenge-response tests.* A challenge-response test is a test involving a set of questions (or "challenges"), that the other entity has to answer in order to pass the test. If the entity provides a satisfactory response to the challenges then it is deemed that the entity has passed the test. The question often relies on the possession of a secret of some sort. A simple example challenge is asking for a password, and the adequate response is the correct password.

The software structure adopted to realize this mechanism adopts the Proxy [16] design pattern. Indeed, the RoomAccessor provides a factory for room proxies. For each existing room there is exactly one corresponding RoomAccessor registered in a remote lookup registry or service. The RoomAccessor's checkAnswer() method provides clients of the room it represents with an appropriate RoomSecurityProxy depending on how the challenge is solved.

The RoomAccessor's getQuestion() method returns an instance of a Question implementation class. One can see on Figure 9 that the framework offers two default kinds of questions represented by two[8] lightweight classes: *(i)* EmptyQuestion is an empty implementation of the Question interface whose execute() method simple returns null; *(ii)* PasswordQuestion represents the simple challenge asking for a password. It fulfills its task by invoking the getPassword() method of the solver it receives as parameter.

The framework also contains a Solver class, which contains one method per challenge supported by the security system. This class simply provides dummy implementations of each method, i.e. simply returning null. This class is intended to be refined and some methods overridden in order to provide correct solutions to the proposed challenges. Typically the avatar will need a smart Solver class which either asks the human user to type a password or provides the solution of the question autonomously.

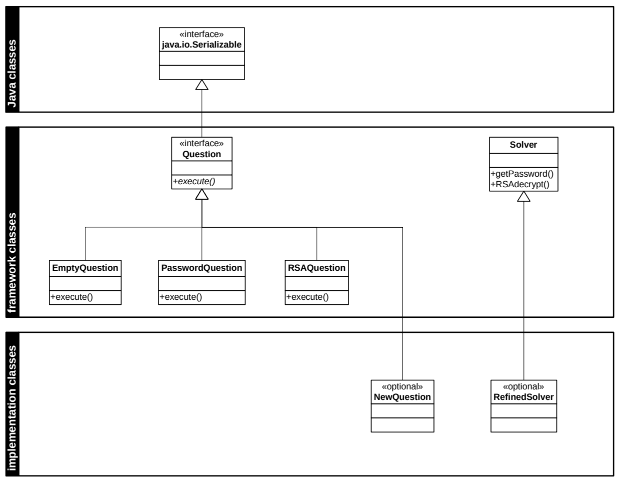


Fig. 9: Challenge-response classes relationships

The sequence diagram of Figure 10 illustrates in greater detail the different steps an avatar has to pass to gain access to a room. The room accessor sends a Serializable Question to the avatar. The avatar locally solves the question through its Solver and receives an answer. The answer is serialized and sent back to the room accessor, which can check it for correctness and create a proxy for the room with the corresponding access rights. This proxy is actually a remote-secure proxy for the room. It is returned to the avatar, which now has a handle for the room.

Note that the communication channel between the avatar and the room accessor may not be secure and some malicious individual could intercept the answer sent by the avatar. Thus sending a password in plain text over this channel clearly represents a security hole. To thwart such kind of attacks a more sophisticated challenge-response must

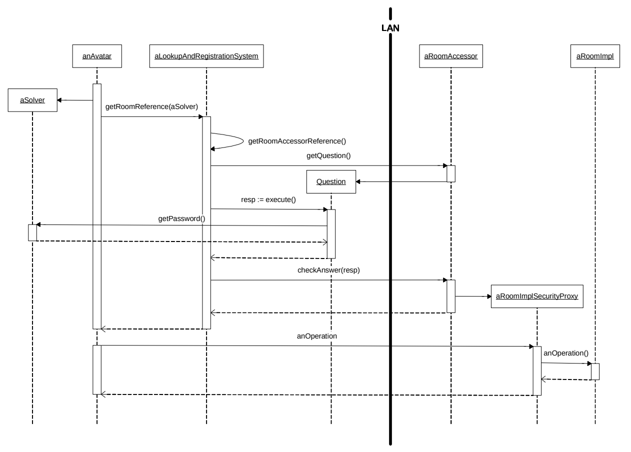


Fig. 10: An avatar getting a secure room proxy

be proposed. An asymmetric (public key - private key) cryptographic algorithm like RSA[9] could be employed to achieve this goal.

Enhancing the MaDViWorld framework with such a new authentication process can be done in two simple steps: *(i)* add a new method to the Solver which could be named RSAdecrypt() and *(ii)* provide a corresponding subclass of Question, for instance RSAQuestion. The new RSAdecrypt() method should be able to manage a key ring to successfully pass the challenges proposed by the different rooms.

Because the security is a difficult topic that may require some experimentation to get right, the security policy of a room is centralized in a single subclass of Question. This allows the framework user to easily try different policies if the existing proves inadequate. Another benefit of the explained architecture is that each room manages its security policy independently allowing for a completely distributed implementation with no central security authority. At installation time, the user who creates the room can choose and parameterize its security policy. Thus we have a simple, yet flexible and powerful security model.

4. MORE ON OBJECTS

The MaDViWorld framework provides a default implementation both for a simple avatar application (see Figure 2) and for the room server application. These two default applications allow for a given amount of customization from their users (e.g. by defining the rooms' access rights and security policy, by setting them up with one's own collection of objects or by connecting them through doors). It would even be possible for an experienced Java programmer to use the carefully designed hooks of the framework in order to extend or even to fully override the default implementation, introducing for example rooms with 2D or 3D representations.

It is, however, not the main goal of the MaDViWorld project

[8]In fact three subclasses are depicted but the RSAQuestion class is not part of the framework. It will be discussed later.

[9]The RSA algorithm was first described in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman [29]; the letters RSA are the initials of their surnames. The interested reader can find a comprehensive discussion of this algorithm in [22].

to go into this direction. We truly believe that the actual implementation is sufficient in order to design exciting worlds at the important condition of disposing of a rich enough variety of objects to populate them. This is the reason why we concentrated on facilitating as much as possible the process of programming new types of object with the ultimate goal of instigating a rich community of object creators. Furthermore, in order to bootstrap this process, we launched a series of student projects[10] with the only requirement of validating the framework by programming new "useful" objects. The next subsections briefly present some of them.

Resource Sharing Objects

The objects are executed either on the machine hosting their containing room or on the one where the avatar application is running. This allows for *resource sharing*. Objects needing a lot of computing power and memory are put in a room hosted by a powerful computer and they are remotely controlled by their thin GUIs launched by avatar clients. A little example illustrating this feature is the fibonacci number calculator. Other ones can easily been imagined, for example from mathematical topics such as fractal calculation, cryptography or linear programming solvers.

Collaborative Objects

The MaDViWorld framework offers all what is needed in order to build *collaborative objects*. Indeed, objects can easily be shared by several users and events transparently broadcasted. This allows for the creation of a large variety of objects supporting collaboration among the virtual world users. These objects range from simple shared whiteboards to sophisticated collaborative editors and "chat" utilities. The whiteboard object is an illustrative example from the MaDViWorld programming cookbook guide, while prototypes of a simple collaborative editor and of a powerful chat object (see Figure 11) have been realized in two separate projects.

Multi-player games are also part of this category of objects. Existing examples of multi-user games are the "battleship" game (see Figure 2), the "tic-tac-toe" game, the "minesweeper" game[11] and even a complex "Metal Panic" game composed of three complementary *inter-communicating objects* : robot factories, customizable fighting robots and fighting arenas.

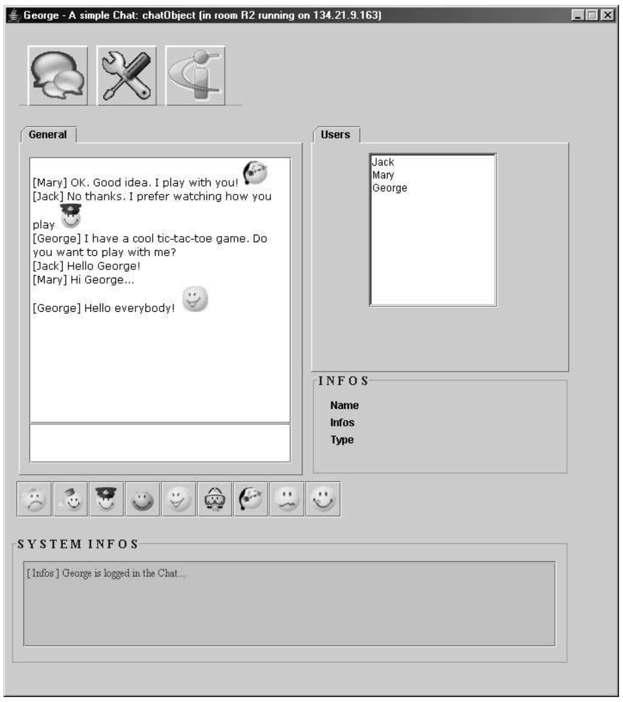
It would also be possible to imagine objects which would sense their environments and adapt their states in order to anticipate the needs of their users, e.g. a whiteboard which would adapt its size to the number of avatars present in a given room.

Inter-Communicating Objects

The remote event mechanism model can also be used in order to make objects communicate with each other. A possible application consists in producing so called "social" objects. For example, one can create a virtual pets community.

[10]Bachelor or Master level projects realized at the DIUF (see [12]).

[11]Essentially a single-user game. It might, however, range in the collaborative objects category if one considers the avatars watching how someone else plays.


George - A simple Chat: chatObject (in room R2 running on 134.21.9.163)

General

[Mary] OK. Good idea. I play with you!
[Jack] No thanks. I prefer watching how you play
[George] I have a cool tic-tac-toe game. Do you want to play with me?
[Jack] Hello George!
[Mary] Hi George...
[George] Hello everybody!

Users

Jack
Mary
George

INFOS

Name
Infos
Type

SYSTEM INFOS

[Infos] George is logged in the Chat...

Fig. 11: A chat object

The avatars owning these pets have to play with them, clean or feed them in order to keep them healthy. If a member of the community dies, the other pets living in the same room are affected by the death of their friend and their "life capital" decreases. The GUI of such an object is illustrated by Figure 12.

Other applications of the communication between objects are the robots, factories and arenas of the "Metal Panic" game, and MadTunes, an audio player accessing MusicRack objects containing several music files.

Agent Objects

The MaDViWorld framework also allows for the creation of so-called *mobile software agents* (see [9, 3]). At the software engineering level (design, programming and especially debugging), these objects are some of the most difficult to deal with. Nevertheless, two prototypical agents have successfully been developed with the help of the framework facilities: The first one is an agent called "Explorer" that draws a sophisticated interactive map[12] of a given virtual world by visiting all its rooms. The second one is an agent called "Matchmaker" that fixes meeting with other agents of the same type on behalf of their respective owners (see Figure 13).

Summary

Our experience within the various projects partially described in the preceding subsections proved that it is rather simple

[12]The map appears as a graph, where the vertices express either the room server hosts, the rooms, the connected avatars or the objects in the rooms, while the edges represent either inclusion (e.g. an object in a room) or connection relationships (e.g. rooms linked by a door).

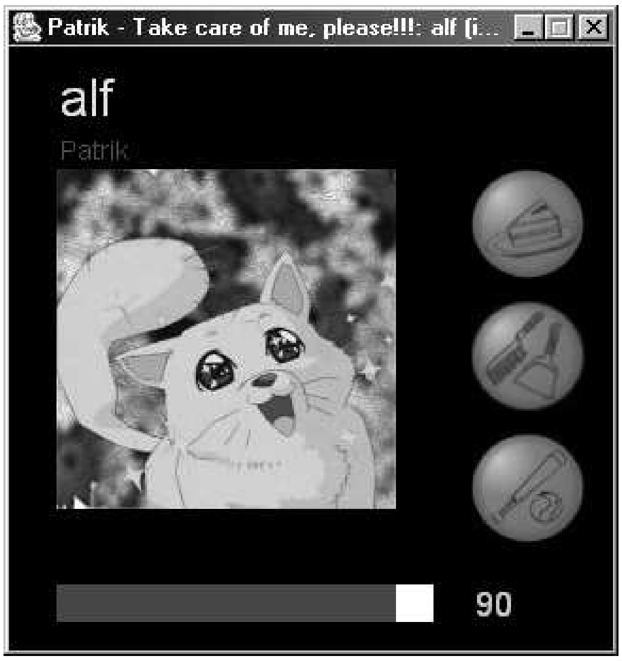
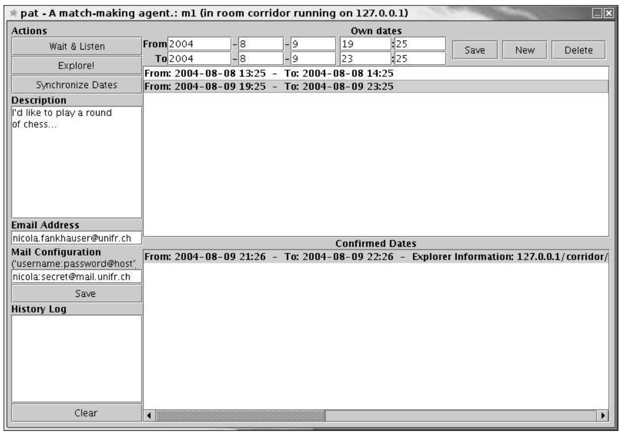


Fig. 12: A virtual pet in MaDViWorld

for an average Java programmer using the framework to develop her own objects with the freedom of deciding if her new object will:

- be stateful or stateless, determining if the internal state of the object is carried around when the objects moves or not;
- be single- or multi-user;
- take advantage of the distributed event mechanism in order to "inter-communicate" with other objects or not;
- take advantage of their intrinsic mobility in order to behave as mobile agents or not;



pat - A match-making agent.: m1 (in room corridor running on 127.0.0.1)

Actions

Wait & Listen
Explore!
Synchronize Dates

Own dates

From 2004 - 8 - 9 19 : 25
To 2004 - 8 - 9 23 : 25

Save New Delete

From: 2004-08-08 13:25 - To: 2004-08-08 14:25
From: 2004-08-09 19:25 - To: 2004-08-09 23:25

Description

I'd like to play a round of chess...

Email Address

nicola.fankhauser@unifr.ch

Mail Configuration ('username:password@host')

nicola:secret@mail.unifr.ch

Save

Confirmed Dates

From: 2004-08-09 21:26 - To: 2004-08-09 22:26 - Explorer Information: 127.0.0.1/corridor/

History Log

Clear

Fig. 13: Screenshot of a "Matchmaker" agent

- be rather specific (for example a given game) or very generic (e.g. a chat, a witheboard or a collaborative editor, which could then be installed by default in every room).

5. CONCLUSION

Designing an extensible and truly decentralized software platform able to support a virtual community based on the MOO paradigm represents a very challenging task at the fringe of today's software engineering technology. In this paper, we have drawn from our experience developing MaDViWorld in order to propose a coherent set of solutions to some of the main questions one must answer in order to embark on such a daunting task. Indeed, the actual version of MaDViWorld is a fully functional framework for creating highly distributed virtual worlds. It has been carefully designed in order to facilitate its enhancement either by extending some of its concrete classes or by implementing the well-documented interfaces of its higher levels.

Although MaDViWorld default avatar and room server applications remain rather simple (i.e. no immersion into 2D or 3D spaces), we truly believe that the actual implementation is sufficient in order to design exciting virtual worlds by creating a rich enough variety of objects to populate them. This is the reason why we concentrated on facilitating as much as possible the process of programming new types of objects.

Our experience with various projects proved that it is rather simple to develop new objects and to test them in a MaDViWorld, with the transparent additional advantages of mobility, remote execution and persistence.

The next step would be to integrate the work already done within a coherent and "interesting" world. Two possible candidates are sketched below: the first one ranges in the area of entertainment and the second one deals with e-learning.

Gameworld

This virtual environment consists of a set of rooms full of active collaborative game objects, ranging from single user arcade games to sophisticated multi-user ones (card games for instance). After having paid a fee, the users are allowed to visit the rooms; to watch other users play; to try out some demo versions of the games; or even to join a game and to exchange their impressions about it. Later, if she is interested, a user can even copy a given game object onto her own machine by getting the right to clone it. A slightly modified version of this world would be to replace the cloneable game objects by active pieces of art that would be unique in the sense that one could only move them around, not copy them.

Eduworld

A more ambitious project is to build up a *distributed learning environment* on the top of the MaDViWorld framework. While Figure 14 sketches the conceptual model of such a world, its key elements are enumerated below.

- *Individual professors' offices* are used in order to re-

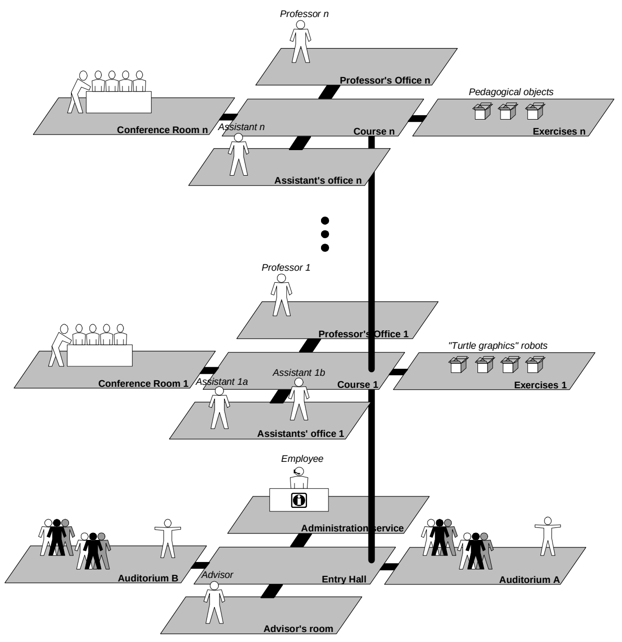


Fig. 14: Distributed learning environment conceptual model

ceive students for private discussions. We propose to physically decentralize them on the professors' private machines.

- *Assistants' offices* are rooms used by the assistants of a given professor in order to receive individual students for questioning about their on-going homework. The functions of these rooms are close to the former ones and we also propose to decentralize them.

- *Conference rooms* are associated to a professor's group and are used by both the professor and his assistants in order to have an open discussion with several students at once. They can also serve for more classical *ex cathedra* courses. These rooms can either be decentralized on a machine associated with a given professor's group or put on a larger department's server.

- *Exercises room*s are the most interesting ones, since they contain the *active pedagogical objects* associated with a given course. For instance, programmable drawing robots could be used in order to teach algorithmic concepts. This idea is analogous to the turtle graphics methodology adopted by Logo [25, 1]. Adapted to a virtual world environment such a learning strategy would lead to the following scenario. Each student clones the 'exercise of the day' robot and takes it into her virtual office, running on her own physical machine. She then tries to instruct the robot to do a given drawing. Once she is finished, the student puts her programmed robot in another room for correction (the assistants' office for instance). A reasonable solution is to put these rooms on the same server as the conference ones. They will not overload this machine, since the real work will always take place on the students' individual machines.

- Administrative rooms provide various central services (registration, accreditation, etc.) and would typically run on a larger department (or even university) server.

It is our hope that such worlds will be built in the near future.

6. REFERENCES

[1] H. Abelson and A. A. diSessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics.* MIT Press, September 1986.

[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] J. M. Bradshaw. *Software Agents.* AAAI Press, 1997.

[4] L. P. Burka. The MUDdex. [online], 1993. http://www.linnaean.org/~lpb/muddex/ (accessed November 26, 2004).

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

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns.* John Wiley & Sons, 1996.

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

[8] V. N. et al. The COVEN project: Exploring applicative, technical, and usage dimensions of collaborative virtual environments. *Presence*, 8(2):218–236, 1999.

[9] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages.* Springer-Verlag, 1996.

[10] E. Frécon and M. Stenius. DIVE: A scalable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(3):91–100, September 1998.

[11] P. Fuhrer. *Distributed Virtual Worlds - Abstract Model and Design of the MaDViWorld Software Framework.* PhD thesis, Department of Informatics, University of Fribourg, Switzerland, Nr. 1458, September 2004.

[12] P. Fuhrer. MaDViWorld (Massively Distributed Virtual Worlds). [online], 2004. http://diuf.unifr.ch/softeng/projects/madviworld/ (accessed November 26, 2004).

[13] 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.

[14] 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.

[15] A. Gachet. *A Software Framework for Developing Distributed Cooperative Decision Support Systems*. PhD thesis, Department of Informatics, University of Fribourg, Switzerland, Nr. 1402, February 2003.

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

[17] C. M. Greenhalgh. Awareness-based communication management in the massive systems. *Distributed Systems Engineering*, 5(3):129–137, September 1998.

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

[19] R. Kazman. Load balancing, latency management and separation of concerns in a distributed virtual world. In A. Y. Zomaya, editor, *Parallel Computing: Paradigms and Applications*. International Thomson Publishing, November 1995.

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

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

[22] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.

[23] B. Meyer. *Object-Oriented Software Construction*. The Object-Oriented Series. Prentice-Hall, 2nd edition, 1997.

[24] K. L. Morse, L. Bic, and M. Dillencourt. Interest management in large-scale virtual environments. *Presence*, 9(1):52–68, 2000.

[25] S. A. Papert. *Mindstorms: Children, Computers and Powerful Ideas*. Basic Books, 2nd edition, March 1999.

[26] Paradise project web site. [online]. http://www.dsg.stanford.edu/paradise.html (accessed January 14, 2004).

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

[28] E. Reid. *Cultural Formations in Text-Based Virtual Realities*. Masters thesis, English Department, University of Melbourne, January 1994.

[29] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):110–126, February 1978. Previously released as an MIT "Technical Memo" in April 1977, [Retrieved December 16, 2004, from http://theory.lcs.mit.edu/~rivest/rsapaper.pdf].

[30] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.

[31] J. Smed, T. Kaukoranta, and H. Hakonen. A review on networking and multiplayer computer games. Technical Report Technical Report 454, Turku Centre for Computer Science, April 2002.

[32] B. Venners. How to attach a user interface to a jini service: An in-depth look at the serviceui project from the jini community. *JavaWorld How-To-Java*, October 1999. [Retrieved December 16, 2004, from http://www.javaworld.com/javaworld/jw-10-1999/jw-10-jiniology.html].

[33] B. Venners. *The ServiceUI API Specification (Version 1.1)*. Artima Software, October 2002. [Retrieved December 16, 2004, from http://www.artima.com/jini/serviceui/Spec.html].

[34] R. C. Waters, D. B. Anderson, J. W. Barrus, D. C. Brogan, M. A. Casey, S. G. McKeown, T. Nitta, I. B. Sterns, and W. S. Yerazunis. Diamond park and spline: Social virtual reality with 3D animation, spoken interaction and runtime extendability. *Presence*, 6(4):461–481, August 1997.