SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2002; **32**:645–668 (DOI: 10.1002/spe.453)

MaDViWorld: a software framework for massively distributed virtual worlds

Patrik Fuhrer*,†, Ghita Kouadri Mostéfaoui and Jacques Pasquier-Rocha

Department of Informatics, University of Fribourg, Switzerland, Site Regina Mundi, Rue P.A. de Faucigny 2, CH-1700 Fribourg, Switzerland

SUMMARY

The MaDViWorld project represents an original attempt to define an appropriate software architecture for supporting massively distributed virtual world systems. A non-massively distributed virtual world system is typically engineered as a client–server application for which a single server or more rarely a small cluster of servers contain all the world pertinent data and assume the world accessibility, consistency and persistence. On the client side, many of them enable interaction with the other users and the various objects of the world. The main originality of our approach resides in the fact that the server part of the proposed system is no more limited to a few centralized servers, but can be distributed on arbitrarily many of them. Indeed, MaDViWorld, the prototypal software framework already implemented using Java and RMI by our group, allows for creating the rooms of a given world on several machines, each running the server application. It is then possible to connect the rooms by way of simple doors and to populate them with active objects. Finally, avatars managed by the client application visit the rooms and interact with the active objects either directly on the remote host or locally by cloning or transporting them first to the client machine. This paper draws from the experience gained with the development of our first prototype in order to discuss, both at the user's and the vi-world developer's level, the main software engineering issues related to the implementation of such massively distributed virtual world systems. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: virtual worlds; MUD; MOO; distributed software architecture; Java; RMI

1. INTRODUCTION

In today's Internet technology, the distinction must be made between applications based on a document paradigm versus those based on a virtual world paradigm, as follows.

- Within the *document paradigm*, documents, often active ones being able to react to various user actions, are made available on one or several servers and client applications (e.g. Web browsers)

*Correspondence to: Patrik Fuhrer, Department of Informatics, University of Fribourg, Switzerland, Site Regina Mundi, Rue P.A. de Faucigny 2, CH-1700 Fribourg, Switzerland.
†E-mail: patrik.fuhrer@unifr.ch

Received 23 July 2001
Revised 20 December 2001
Accepted 7 February 2002

can be used in order to interact with them. Typically, each user copies the documents onto 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 really massively 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 theoretical 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 (e.g. the same room) must immediately be made aware of it. Examples of the virtual world paradigm range from simple graphical chat to sophisticated 3D virtual worlds used for military simulations (see Section 2.2).
  At the software architecture level, systems based on the virtual world metaphor are clearly the more complex ones. Indeed, the users directly interact 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. Therefore, most of them are based on a software architecture with one central server containing all the world pertinent data and assuming the world accessibility, consistency and persistence and many clients allowing interaction with the other users and the various objects of the world. This approach has two main weaknesses. First, the whole system depends completely on the central server robustness. Secondly, it does not scale well. Just imagine what the World Wide Web would be today if all its contents would need to be regrouped on a single server having to coordinate all of its users.

The goal of our research group is to define and test promising software solutions in order to support the virtual world paradigm presented above, without making concessions to the single server architecture. Actually, *MaDViWorld*, the acronym of the prototypal software framework presented in this paper, stands for Massively Distributed Virtual World, since its subspaces (e.g. rooms) 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 the Internet. This obligation is exactly the same as the one required by the World Wide Web with all its advantages in terms of robustness and scalability. Note that the definition of prototypal architectures for really distributed virtual worlds is a new active field of research tackled both by

[‡]An indirect outcome is, however, possible by means of techniques such as forms, CGI and servlets. For example, a document that is generated on the fly by a program running on an http server depends often on data fetched from a database through HTML forms by its users.

the vi-world community and the distributed software community. An interesting attempt to build a highly distributed system has been made recently by the V-Worlds Group at Microsoft research as an enhancement to their V-Worlds platform (see [1,2]). Each server manages a room and its content. An SQL server hosted on one of the world server machines handles the synchronization mechanism. Other proposals for completely distributed environments can be found in the following projects: URBI ET ORBI [3], DIVE [4] and MASSIVE [5]. The NOMAD framework (see [6]) proposes an interesting CORBA based approach that encourages the division of a vi-world into separate locations, each running on its own server. Nevertheless, it still needs a central server for coordinating the whole world. Another approach focused on the use of the CORBA platform as a middleware layer to support distributed vi-worlds can be found in [7]. The latter, however, is based on a 'single server—many clients' architecture.

As pointed out in the previous paragraph, this paper investigates possible solutions for the creation of a highly distributed virtual world software framework. Furthermore, it concentrates on our MaDViWorld prototypal implementation for illustrating the proposed solutions. In order to best achieve this goal, the paper is structured as follows. Section 2 presents a brief definition and a historical overview of the virtual world metaphor and introduces the reader to the accompanying terminology, i.e. avatars, rooms and objects. Naturally, the overview part of the section is very much focused on the goals of this paper and thus has not the ambition of being exhaustive. Section 3 summarizes, both at the user level and at the vi-world developer level, the technological objectives that were the motivation behind the project of creating the MaDViWorld framework. Section 4 concentrates on the dynamic behavior of vi-worlds implemented on top of the MaDViWorld framework. It uses a simple example in order to provide the reader with a feeling of how the mapping between a user's mental model of her vi-world and its concrete 'physical' realization takes place. Section 5 presents an overview of the static software architecture of the current[§] version of the MaDViWorld framework by sketching the three layers of its class hierarchy. This section is the most technical of the paper and represents its true contribution. For the interested reader, Section 6 of [8] presents the same material at a more detailed level. Finally, Section 6 enumerates the main achievements we have drawn from our work, while Section 7 contains some suggestions for improvements at various levels.

2. VIRTUAL WORLDS: A BRIEF PRESENTATION

2.1. Terminology

In [9], *virtual worlds* are defined as 'computer-based models of three-dimensional spaces of objects with restricted interaction'. It is further asserted that *multi-user worlds* are distinguished by the fact that several users (working on different machines) can move through the world and interact with one another or with the objects at the same time. Finally, a virtual world is considered as *distributed* 'if active parts of it are spread throughout different computers in a network'.

The present article respects the above definitions on the condition of giving a rather permissive interpretation to the expression 'models of 3D spaces'. Indeed, we consider that a computer-based

[§]The 1.3 beta version.

model of a given world should not necessarily respect a precise 3D, or even 2D topology. In a text-based model, for example, it is not necessary to be precise about the coordinates of avatars and objects present in a given room. A simple list that is automatically updated each time a change occurs in the room and which allows one to interact with the user or the object of one's choice, might represent a suitable model for many purposes. The same is true for doors. It is certainly more realistic to place them in a precise location within a 3D or 2D wall and to force the user to virtually open them in order to move into another subspace. It is, however, also possible to present a list of available doors in a given room and to automatically transfer the user in a new context (i.e. new room, with new users present and new objects available), when she selects a given door. To summarize, we believe that the feeling of being in a space with other users and objects can be achieved within a very simple topological model (by only defining rooms, for example). Naturally, the underlying architecture of the world should be conceived in such a way that the addition of a more realistic 2D or 3D topology remains a possible option.

For a good comprehension of the present paper, the following three terms need to be explained in more detail.

1. *Avatars*. Diehl [9] defines them as the 'virtual representations of the users'. He then explains various perspectives from which the user can look at the scenery (i.e. through the eyes of the avatar, with a camera on top of it or with a camera behind it) and he finally states that in multi-user worlds an avatar should also somewhat be the visual representation of its user.
   Within this paper, we adopted the following close but nevertheless easier to apply definition. 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. This definition does not contradict that of Diehl, but it also enables an avatar to be a simple client text-based application broadcasting the actions of its user through simple messages to the users of avatars sharing the same subspace.
2. *Rooms*. When a virtual world does not have a precise topology, it is very difficult to distinguish between near and distant elements. Thus, it is essential to divide the world into subspaces where the users might or might not enter and in which all interactions take place. Otherwise, the world would not scale. In the present paper, we call such subspaces rooms, and an avatar can move from one room to another either by using the *doors* available in a given room or by direct transfer if she knows the address of the destination room.
3. *Objects*. In [10], the author distinguishes between three categories of objects as follows.
   (i) *Passive objects*, which can only change if a human agent (through an avatar application) interacts with them. These objects do not react to changes in other objects. A simple whiteboard on which each user can write short messages is an example of such an object.
   (ii) *Reactive objects*, which can change their states in response to changes in other objects. These objects typically obey 'physical' laws of interaction. When hitting a wall, for example, a ball will rebound at a given angle and velocity.
   (iii) *Active objects*, which can transform themselves. A whiteboard, which would adapt its size to the number of participants in a room, is such an object. An active object has a minimal intelligence built into it.

   We strongly think that a virtual world architecture should allow all three types of objects mentioned above and that this can be achieved by modeling them as instances of software

objects (i.e. the combination of a state and of methods that can be executed by the host machine). Furthermore, in a distributed world, it should be possible to physically transport a given object from a room on machine X to a room on machine Y (see Section 3).

2.2. Historical overview

The history of virtual worlds began with the first *MUDs* (Multiple-User Dimensions or Dungeons) designed as role-playing adventure games in the mid-seventies. A MUD is a shared textual virtual environment that can be explored by its users through a series of simple commands such as 'look', 'go' and 'take'. As she moves within the world, a user runs across other users with whom she either chats, collaborates for a given task or fights for resources. A detailed description of MUDs is given in [11] and a huge amount of well-structured information and resources on actual MUDs (some having evolved from a textual based representation to becoming sophisticated 2D or 3D models) can be found on the Website [12].

In 1990, object-oriented features (i.e. the possibility to consider the entities of the world as instances of programmable software objects) were added to the MUDs, which were renamed *MOOs* (MUDs Object Oriented). *LambdaMOO* was the first and was officially opened at Xerox PARC by Pavel Curtis in 1991 (see [13]). It was designed for social gatherings instead of role-playing. Note that each MOO (or MUD) has its own creator (or God) who sets up all its rules and topology. Thus, it is always entirely contained and managed by a single central server where the users can connect through more or less sophisticated client applications. In that sense, MOOs are interesting for us from an end-user point of view, but their software architectures present very little appeal in terms of robustness and scalability. They are the opposite of being distributed. This limitation holds true for the first 2D and 3D chat worlds, which appeared on the Internet around 1995, as well as for the first multi-user games and VRML-based virtual worlds. An interesting listing of such applications, as well as an introduction to their terminology and principal techniques (e.g. VRML[¶]), can be found in [9].

More recently, Microsoft developed its V-Worlds platform (see [14]) on top of COM, which is still based on the concept of a central server, but plans exist to make it really distributed (see [2]). The direction of research taken by the Microsoft V-Worlds group is typical of the most recent trend in virtual worlds research. Researchers from the software engineering and distributed architecture communities try to apply technologies such as DCOM, CORBA and Java to tackle the difficult problem of defining standard protocols and of creating extensible software frameworks for supporting distributed virtual worlds. We have already mentioned some of these projects in the introduction. An interesting taxonomy of the problems one may encounter when dealing with large networked virtual environments can be found in [15].

3. TECHNOLOGICAL OBJECTIVES

Section 2 has clearly demonstrated that the creation of an appropriate software environment for supporting the development of multi-user distributed virtual worlds is both a new field of research

[¶]Virtual Reality Markup or Modeling Language. At the end of 1997 a revised version VRML97 became an official ISO-standard.

and a very ambitious task. Thus, if one does not want to be overwhelmed by the complexity of the work, one must absolutely follow a feasible strategy based on a few well-defined technological objectives, which will be considered as having first priority. In the next two sections, we will state these objectives more precisely in the context of the MaDViWorld framework. Section 3.1 considers the point of view of a simple user (i.e. a person who explores and interacts with a world developed on top of MaDViWorld), while Section 3.2 concentrates on the requirements of a system developer (i.e. one who is ready to do some real programming in order to extend the framework or to create new types of objects).

3.1. The user level

At the user level, the main idea is to keep the operations in the virtual world as intuitive and transparent as possible. In MaDViWorld, this objective is reached by satisfying the following non-exhaustive list of requirements.

- *World topology*. Rooms (as HTML pages for the World Wide Web) can be hosted on many machines worldwide and doors topologically connect them. Avatars access a given room either by specifying its name and the IP address of its host machine or by transparently crossing one of its doors.
- *Interactions*. A soon as she has entered a room, a user is able to interact with other users and/or objects present in it. In principle, the users are informed of her actions.
- *Active/mobile objects*. Initially, each room is populated by a set of objects. These objects can be executed either on the remote machine where they are located or on the local machine of the user. They can also be copied (cloned), removed or simply 'physically' transported by an avatar and moved to other rooms. It is further possible to provide these objects with sophisticated graphical user interfaces, which are always transparently executed on the local machine of the user.
- *World creation and extension*. A set of simple applications (wizards) must allow a user to effortlessly introduce newly programmed objects into the rooms of her choice, as well as to easily extend the global virtual world by creating and managing new active parts (rooms) on any machine running the appropriate server application. Actually, this task is very similar to that of creating new HTML documents and then making them available to other Internet users by running an http server.
- *Persistence and recovery*. The concept of persistence must be implemented. Either a part of the active world (i.e. the one existing on a given machine) or an avatar can be deactivated, its state stored on a local file and then reactivated. This must work both in the case of a voluntary interruption and in the case of a software or hardware failure.
- *Security*. An appropriate security policy must be enforced both at the machine and at the vi-world level. In the first case, it means that the operating and file systems of the host machines for the client (avatar) and the server (rooms) applications must be protected from potentially dangerous operations (e.g. through a sandbox model). In the second case, rights within the vi-world itself must be defined and enforced (e.g. not every avatar should be allowed to enter any room, nor to interact with, delete, move or copy any objects).

Note that the possibility of creating 3D worlds and/or using audio or video interfaces, although not excluded, is not on our first priority list.

3.2. The vi-world developer level

At the vi-world developer level, the key factor for success is to propose an open architecture, allowing a knowledgeable programmer to easily extend or ameliorate the current version of the framework. Section 5 describes the technical choices (e.g. adoption of a layered class architecture, separation into well-defined packages, etc.) implied by this objective. The non-exhaustive list below provides the reader with a preview of these requirements.

- The developer who wishes to add *new types of objects* to the world must be able to do so (1) by using a standard technology and (2) by respecting a small set of rules. An acceptable solution would be to require her to use the Java programming language and to offer her a structured set of abstract interfaces and concrete classes that have to be implemented and inherited, respectively, in order for an object to be 'vi-world compatible'.
- Although not trivial, the programming of *a new avatar* should be accessible for an average programmer. This implies that the concrete avatar implementation must be well separated from the other parts of the system (e.g. encapsulated in a Java package) and that the set of minimal conditions it has to respect must be regrouped in a somewhat abstract level (e.g. a Java interface class).
- Finally, the programming of *new types of rooms*, with more features or special behaviors than the standard ones, must also be possible for a developer who is willing to put some effort into understanding the philosophy behind the code of the existing framework. This can be achieved by offering an abstract definition of room properties at the highest possible level. These properties must always be respected (implemented), but they can also be extended or redefined through an object-oriented language inheritance mechanism.

4. THE MaDViWorld DYNAMIC BEHAVIOR

After the general considerations of the preceding sections, it is time to concentrate on our research group's concrete contribution: the MaDViWorld software framework for massively distributed virtual worlds. Before going into the details of the framework's software architecture in Section 5, we focus first on the concrete dynamic behavior of a vi-word based on our technology. In order to achieve this goal, we consider first an example that is at the same time complete and simple. It is complete in the sense that it is sufficient in order to illustrate all the major features of the framework (object mobility, physical distribution of the different parts of the world, avatar interactions, etc.) and remains simple, since it is composed of only two rooms and one active object running on two different machines and since it is visited by only three avatars running on three different machines. The example chosen is not very attractive, but it has the advantage of helping the comprehension of many interesting concepts, without overwhelming the reader. Section 4.2 shows how the physical distribution of such a vi-world is kept transparent to the end-users and discusses the framework's potential for massively distributing

more complex worlds. Finally, Section 4.3 provides the reader with an insight into richer on-going projects based on MaDViWorld.

4.1. A simple virtual world

This section presents a sample user session with a very simple[‖] MaDViWorld-based virtual world. Its purpose is to illustrate some of the main functionalities of our actual prototype, without entering into its more complex technicalities. In order to best achieve this goal, we consider a typical scenario decomposed into seven major steps.

Launching a server and setting up a room

Step 1. Sylvia launches the MaDViWorld server application on her machine. Then she uses the wizard in order to create a new room 'R1' and for adding a new TicTacToe active object in it. She furthermore decides that every other user can enter R1 and play the TicTacToe game.

Launching an avatar

Step 2. By launching the MaDViWorld avatar application, Sylvia enters the virtual world. She chooses an avatar name (actually Sylvia) and specifies the access path of the room she wants to be connected[**] with (i.e. `//134.21.9.252/R1`).

Step 3. Another user, James, launches the avatar application from his machine and enters R1. The avatar application displays all the rooms connected with doors to the current one as well as the lists of all the users and objects in the room.

Using objects

Step 4. James sees that Sylvia is in R1. He invites her to play the TicTacToe game by sending her a message via the chat window (see Figure 1). She accepts and both users launch the TicTacToe user interface by selecting 'TicTacToe' from the objects list and clicking on the 'Start' button.

The game object recognizes the first and the second connected player (see Figure 2) and after each move waits for the right player to take her or his turn. Each game update is automatically reported on both clients.

Step 5. A third user, Hans, also enters R1 and launches the TicTacToe object. Since this game does not accept more than two players, Hans is prevented from interacting with it. He can, however, observe the game evolution.

[‖]It is composed of only two rooms R1 and R2 on two different machines, one simple active TicTacToe object and three users: Sylvia, James and Hans.

[**]It is also possible to access the corridor (i.e. `//134.21.9.252/corridor`), which is the default room on any machine running the server application. This is useful when a client does not yet know the existing rooms.

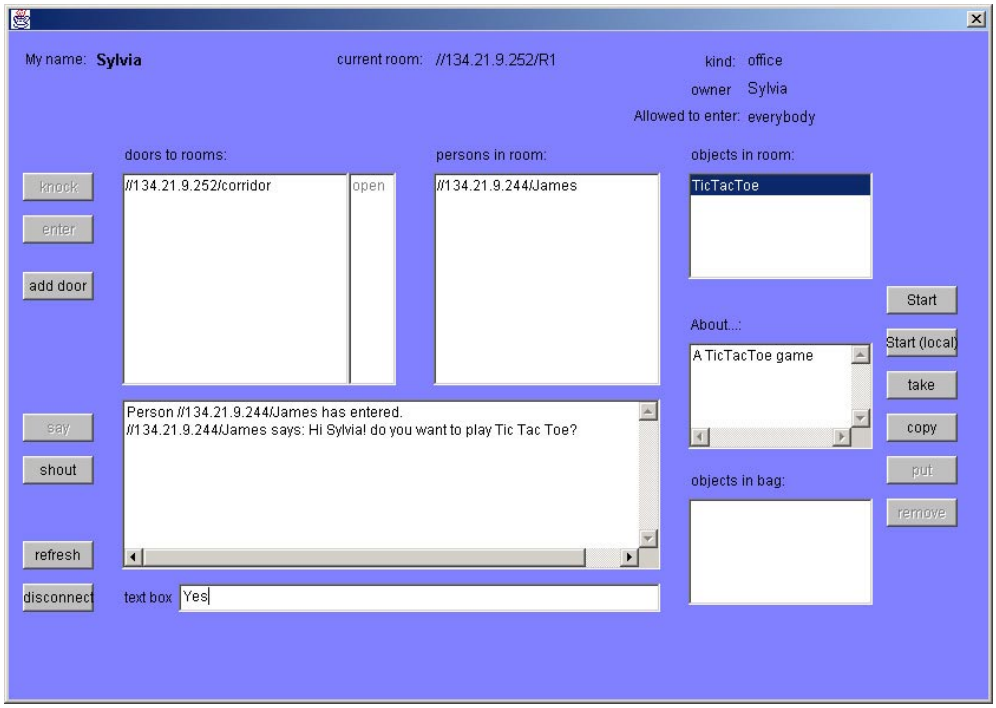

Figure 1. Avatar client application.

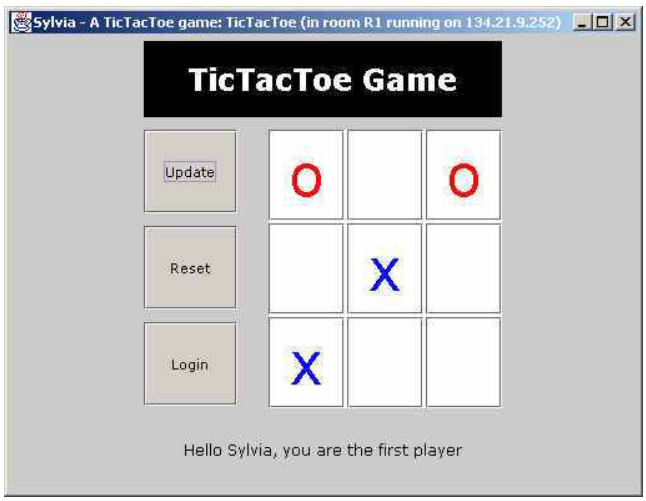

Figure 2. TicTacToe game interface.

Objects mobility

Step 6. Sylvia and James decide to transport the game to another room and play the revenge there. James launches the server application and creates a new room 'R2' on his machine. Then he connects it with R1 by creating a door between the two rooms.

Accessing a room via a door is easy, since each avatar automatically displays each new connected room with the current one.

Step 7. In order to move the TicTacToe game to room R2, Hans selects the object and clicks on 'take'. The object is automatically put into his avatar's bag (see Figure 1). Then he enters R2 and clicks on 'put'. Sylvia joins R2 and both users replay the game. Note that the TicTacToe object has been 'physically' moved from room R1 to room R2 and that it is no longer available in R1. If James had copied it instead of taking it, there would now be two distinct TicTacToe instances, one still in R1 and the other in R2.

4.2. Conceptual versus physical model

The aim of this section is to carry out what is possibly the most important benefit of the MaDViWorld's approach, namely what we call 'massive' distribution. In order to achieve this goal, we must carefully study how the different parts of a MaDViWorld based vi-world are distributed, both from the end-user's perspective and from that of the physical machines running the system.

Let us reconsider the state of the preceding simple example's vi-world at the end of Step 7. The users James, Sylvia and Hans perform their tasks with a fairly simple mental model of the world in their minds: each one's avatar is in a given room, possibly with other avatars; the room is likely to contain objects with which the present avatars can interact; and finally the room most probably has one or several doors, that the present avatars can use in order to move to other rooms. Figure 3 summarizes this vision and can be considered as an appropriate representation of our simple vi-world *conceptual model*. In other words, for the end-user, there are rooms and doors her avatar can go through and she can use the latter in order to explore the world transparently, without knowing on which machine the room her avatar is currently visiting runs on. Note that an avatar can also reach a given room directly under the condition that her user knows its corresponding address[††].

Behind the scene, however, at the concrete level of the computers supporting the system, things are more complicated. As it is graphically outlined within the *physical model* shown in Figure 4, there are two room server applications and three avatar applications all running together on three different machines. Each machine maintains a naming service (`rmiregistry`) for the virtual rooms and the avatars running directly on it. Finally, the relations (methods firing, events distribution, etc.) between the objects contained within a given room and the avatars connected to it are managed by the room itself.

[††]In the current version of the framework, the address consists of the name of the room appended to that of the physical machine it runs on. In Section 7.2, we shall hint how a next version might provide both a simpler and a more powerful mechanism to find such an address.

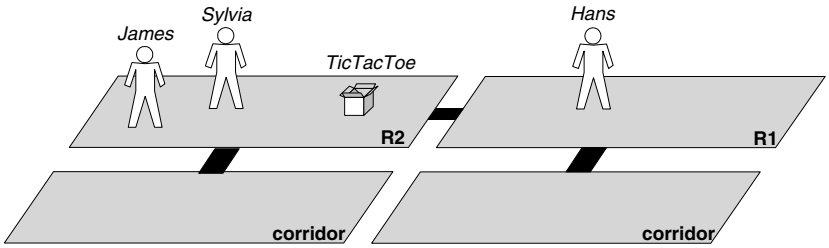

Figure 3. Simple world's conceptual model.

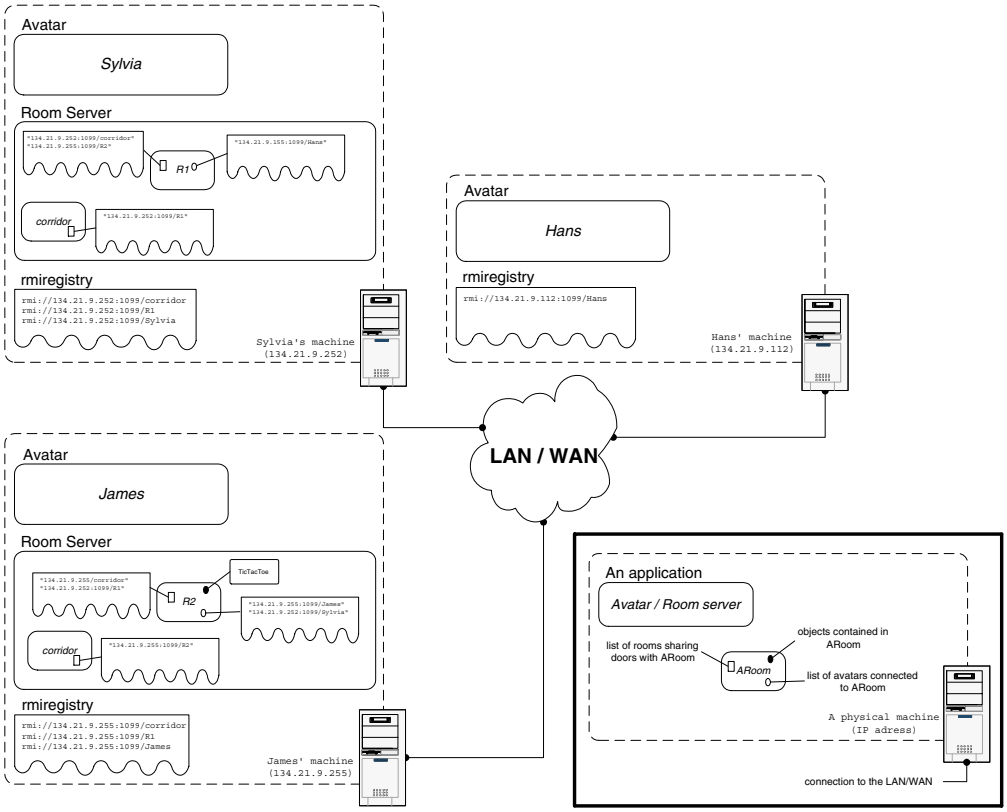

Figure 4. Simple world's physical model.

Naturally, a given room or the RMI naming service of a given machine might become bottlenecks, if one either populates a room with too many objects and avatars or puts too many rooms on a single server. Indeed, MaDViWorld's main strength is not to optimize the interactions of many avatars and objects within many rooms physically present on the same server, but rather to encourage a wide physical distribution of the vi-world resources. We clearly do not encourage the kind of situations where some rooms become central and have too great a relevance for the consistency of the world. On the contrary, we want to support a World Wide Web type of organization, where each individual server only maintains a very partial portion of the whole system. In that sense, we consider that the MaDViWorld framework encourages massive distribution of the vi-worlds based on top of it. Figures 5 and 6 show how the distribution of a more complex example would look both at the conceptual and physical levels.

4.3. Toward richer virtual worlds

As already mentioned at the end of Section 3.1, our framework does not exclude the creation of vi-worlds offering sounds and video, as well as 2D or 3D representations. These aspects, however, are not our first preoccupation and are rather considered as improvements that could be added later within the system implementation layer of the framework (see Section 5.2). We nevertheless consider that even the oversimplified world presented in the last two sections is still rich enough to inspire potential developers to create attractive vi-world applications. Taken from on-going research projects in our department, examples of such applications are briefly proposed below.

- The first one consists of a set of rooms full of *active collaborative game objects*, ranging from single user arcade games to sophisticated multi-user ones (cards 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.
- A more ambitious project is to build up a *distributed learning environment* on the top of the MaDViWorld framework. While Figures 5 and 6 sketch the conceptual and physical models of such a world, its key elements are summarized below.

  - *Individual professors' offices* are used in order to receive 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 homeworks. 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.

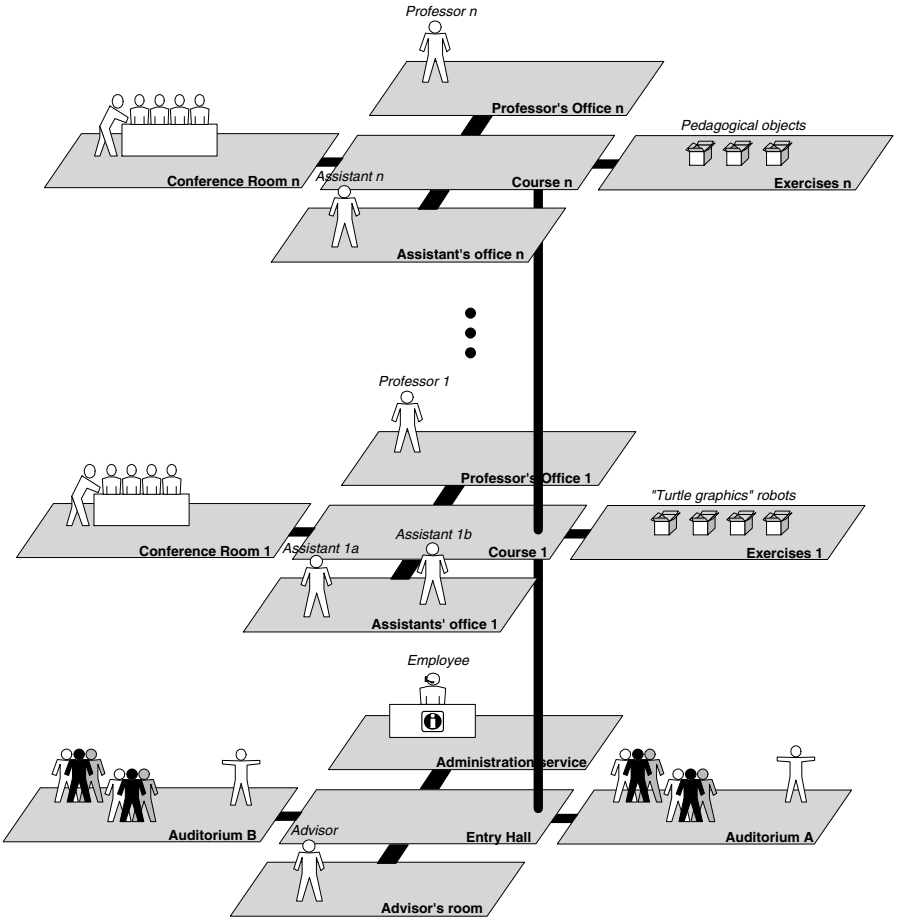

Figure 5. Distributed learning environment conceptual model.

– *Exercises rooms* 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 (see [16,17]). Adapted to a vi-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.

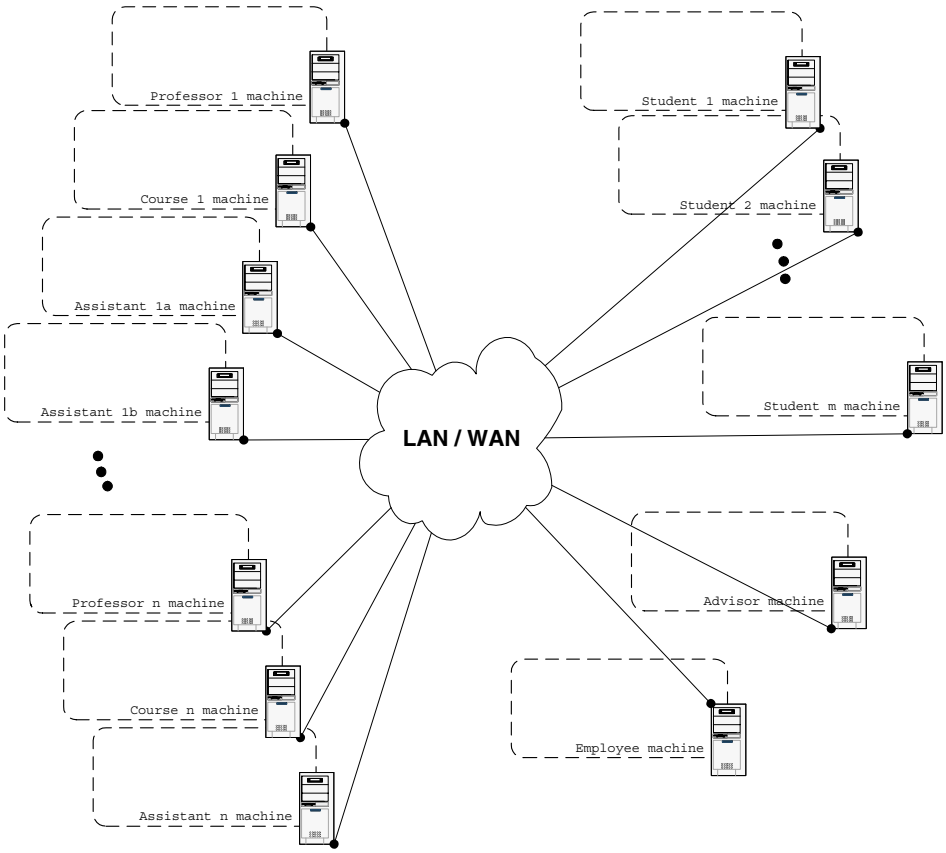

Figure 6. Distributed learning environment physical model.

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.

The interested reader is invited to follow the advancement of such projects on the MaDViWorld official Web site (see Section 6).

5. THE MaDViWorld SOFTWARE ARCHITECTURE

MaDViWorld is a collection of 31 classes organized in eight packages representing more than 5000 lines of Java source code. The MaDViWorld platform was developed with the Java 2 platform

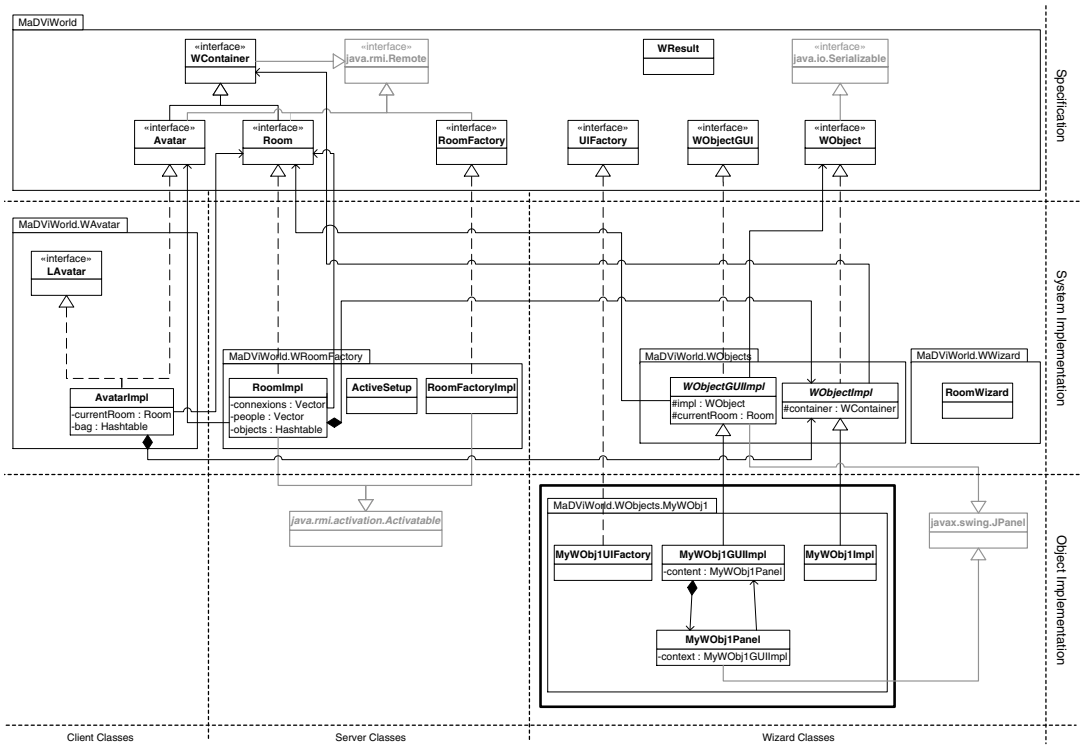

Figure 7. Static UML diagram class of MaDViWorld.

Standard Edition (J2SE™ version 1.3). The eight packages regroup all the classes needed by the three deployment parts of MaDViWorld in order to setup, manage and use a virtual world. In Figure 7, which illustrates the whole‡‡ static diagram of the MaDViWorld framework*, these three parts are separated by vertical dotted lines. From left to right, they correspond to the following independent applications.

- A *client application*, called an avatar, for exploring the world, as well as interacting with its objects and other avatars;
- A *room server*, for creating, hosting and managing the rooms of the world on a given machine;

‡‡In this figure, only six packages can be counted. Indeed, two utility packages used by every deployment part are omitted here for the sake of simplification. The presentation of these packages is out of the scope of this section but can be found in Section 6 of [8].

*The classes in gray are not part of the MaDViWorld framework, but are important Java classes or interfaces that are extended by the framework.

- A *room setup utility*, called a room wizard, in order to remotely setup rooms and put objects into them.

The packages are further distributed into three hierarchical layers, separated by horizontal dotted lines in Figure 7:

- The MaDViWorld *Specification* layer;
- The MaDViWorld *System Implementation* layer; and
- The MaDViWorld *Object Implementation* layer.

This section provides a high level description of the MaDViWorld framework software architecture. The interested reader should refer to Section 6 of [8], as well as to the javadoc documentation of the MaDViWorld official Web site for more information on the precise role played by the many classes and methods of the framework.

5.1. Specification layer

The MaDViWorld specification layer classes shown in the upper part of Figure 7 are illustrated to a larger extent with all their methods signatures in Figure 8. The classes of this layer encapsulate the conventions that each avatar, the room server and the room setup applications must respect in order to function and communicate with one another properly. For example, a room object should offer the names of all the objects it contains (the `getObjectNames()` method) to an avatar and should allow it to add a new one (the `setObject()` method). Ideally, these 64 methods signatures represent the minimal abstract protocol to be implemented in order to create a MaDViWorld compatible virtual world. Actually, this protocol still lacks some features, presents some redundancies and is too implementation oriented. It will be seriously revised in the next version of our prototype. Its main characteristics are summarized below.

- The `Avatar` and `Room` interfaces both extend the `java.rmi.Remote` interface and define all the methods that have to be remotely invoked on an avatar and a room object, respectively. The subset of these methods, which are not concerned with the virtual world objects management, essentially implements a MUD protocol as outlined in Chapter 16 of [18]. Note the necessity of defining a `RoomFactory` interface in order to provide a clean way of installing rooms on the many servers of our virtual worlds.
- The virtual world objects (including the doors special case) are always included within a *container* (i.e. the room containing them or the avatar carrying them). Since it is often vital for an object to know its container and to ask for the host machine it runs on, we have abstracted these features in the `WContainer` interface that both the `Avatar` and `Room` interfaces must extend.
- Within MaDViWorld, the *rooms play a central role*. Indeed, when an avatar wants to invoke a method on a remote object, it must do it through the room containing the object. Reciprocally, when the state of a `WObject` changes, it simply uses its containing room in order to automatically inform all the avatars in its proximity. The object can even force the room to save its state (thus indirectly also the ones of all the objects it contains) by merely setting up the `update` field of the corresponding `WResult` wrapper object to `true`.

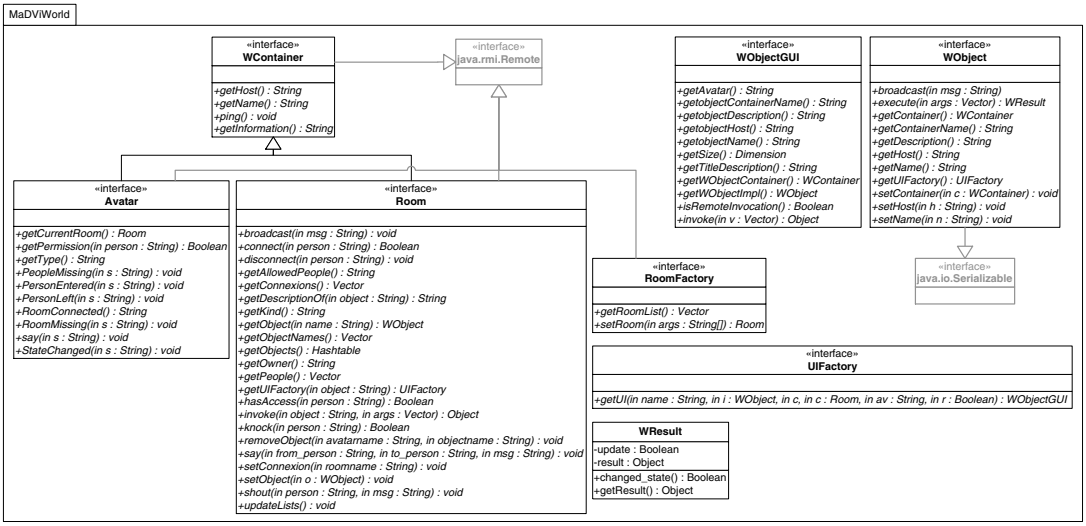

Figure 8. MaDViWorld specification interfaces.

- The objects populating a MaDViWorld-based virtual world have two main properties. First, they are always composed of two well-separated parts: (1) the object itself, which must respect the `WObject` interface protocol; and (2) its user interface, which follows the `WObjectGUI` interface protocol. Further details on how these two parts are installed and on how they work together will be given in Section 5.3. Secondly, they are *mobile objects*. An avatar can take an object from a given room, put it in its bag, go to another room on a different machine, put the object in this new room and run it there. To allow this scenario, objects must be serializable and thus implement the `java.io.Serializable` interface. Within the actual version of the MaDViWorld prototype, object methods are not directly remote invocable. We made the choice of using the rooms containing the objects as relays for that task. Furthermore, a transparent *code transport* mechanism had to be developed in order to support objects and user interfaces mobility.
- Finally, it is important to note that, although not directly enforced by the protocol of this abstract specification layer, it will be necessary, at the implementation level, to focus on providing a *robust persistence mechanism*. The first and second parts of Section 5.2 present our solution to this problem for the client (avatar) and the server (rooms and objects), respectively.

5.2. System implementation layer

This section is divided into three parts corresponding to the vertical separations of the system implementation layer shown in the mid section of Figure 7. Note that the client, server and wizard

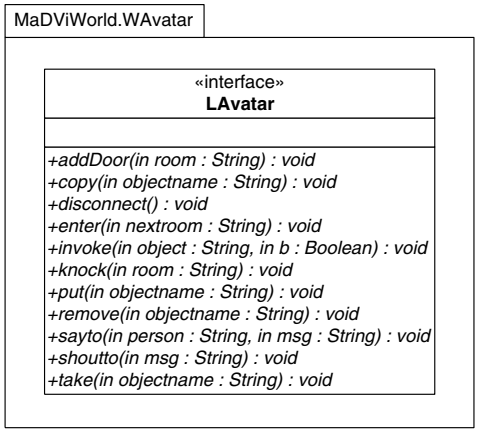

Figure 9. The MaDViWorld.WAvatar.LAvatar interface.

classes presented below represent only *one possible implementation among many* of the ideas that were presented in Section 3 and then clarified by the specification layer protocol of Section 5.1. Our goal is that any serious vi-world developer should be able to rewrite (or rather expand) these classes, by simply respecting a small set of well-established rules.

Client classes

The `MaDViWorld.WAvatar` package contains the `LAvatar` interface and an implementation class that, together with some graphical helper classes not shown in Figure 7, form the MaDViWorld client avatar application. The latter could be seen as a remote console showing the virtual world through the eyes of a virtual person (avatar) controlled by a human user. Note that the `AvatarImpl` class concretely implements both the `Avatar` and the `LAvatar` interfaces. The former is part of the MaDViWorld specification layer and has to be implemented imperatively in order to allow the avatar to exchange vital information with other avatars and rooms. The latter contains methods that we strongly recommend in order to implement a workable avatar. These methods are self-explanatory and their signatures are summarized in Figure 9. In opposition to the methods of the `Avatar` interface, they cannot be invoked remotely. Naturally, the look and feel of the user interface, as well as the way these methods are concretely implemented, are the result of a lot of developers' choices. Therefore the `AvatarImpl` class could be greatly customized or extended by a knowledgeable MaDViWorld developer. For example, within our actual prototype, we have made `AvatarImpl` instances serializable. Thus, a user can save the state of her avatar and reactivate it whenever she wants.

Server classes

The classes of the server layer are bundled in the `WRoomFactory` package and are used in order to support the creation and the management of a virtual world active part (i.e. the rooms and indirectly their contents). It is important to notice that the rooms are distributed on many machines each running the room server application.

A room creation occurs when a wizard application asks a `RoomFactoryImpl` object to create a proper `RoomImpl` instance. Although these two objects must fully respect the protocols defined for them within the specification layer, they are both highly adaptable. Indeed, (1) the `setRoom()` method of the `RoomFactoryImpl` class takes an array of strings as an argument, which basically puts no restriction on the parameters that can be applied to set up a room; and (2) every system developer is free to make her own choices in specifying the way the protection and permission mechanisms of a room work by implementing her own version of critical methods such as `hasAccess()`, `knock()`, `removeObject()` or `setObject()`.

Within our actual prototype, we have made `RoomFactoryImpl` and `RoomImpl` instances activatable and we have used the Java activation model in order to provide a robust server persistence mechanism. This explains the presence of the `ActiveSetup` class in the `MaDViWorld.WRoomFactory` package. For general considerations on the Java remote object activation framework, the interested reader will find [19–21] worth studying.

Wizard classes

The role of the wizard layer classes is to allow users to install their own rooms with their own custom objects on a previously known remote server machine. Within our actual prototype an *ad hoc* RoomWizard graphic application collects the required information from a human user and then invokes the `setRoom()` method of a remote RoomFactory object, as well as the `setObject()` method on the newly created room with an instance of the custom object class to be added as argument. Note that the server needs no prior knowledge of the objects that will be installed in its room, except that they all implement the `WObject` interface. Each of these custom objects will also have to provide a graphical user interface, for which the only restriction consists in implementing the `WObjectGUI` interface. The concrete implementation of the objects and of their graphical user interfaces represents the topic of the next section.

As this wizard application has no other constraints to respect, it offers a lot of extension possibilities in the way the user interface is presented and in the way the pre-programmed objects are made available to the users.

5.3. Object implementation layer

It is our main intent that any knowledgeable Java programmer might create new classes of objects and then populate our virtual world with their instances. In order to gain a large community of such users, we made this process both very simple and as little constraining as possible. Suppose a programmer wants to add instances of a new class of objects named `MyWObj1`. She must merely respect the following strategy: (1) she creates a `MaDViWorld.WObjects.MyWObj1` package as shown in the bottom right part of Figure 7; (2) once the newly created package has been successfully compiled

on her own machine, she uses the wizard application in order to create a local instance of the new object type; and (3) she once again uses the wizard to install the newly created instance in any local or remote room she may have access to. Other users can then copy and/or move this original instance, thus populating the world with many copies of it, each having its own identity and state. Note that the `MaDViWorld.WObjects.MyWObj1` package is always composed of the four following classes, each having a clearly defined role as follows.

- The `MyWObj1Impl` class implements the concrete methods that the object will be able to execute.
- The `MyWObj1Panel` class contains the *graphical user interface*, which is totally separated from the object implementation. We strongly suggest that this class extends the standard `javax.swing.JPanel` class. This way, it will be possible to directly edit its graphic components with any standard Java IDE[†], like Forte™.
- The `MyWOjb1UIFactory` and `MyWObj1GUIImpl` classes are lightweight classes, which could be automatically generated. While the first one simply delegates the job of concretely creating the graphic user interface to the `MyWObj1Panel` class, the second implements the `getUI()` method of the `UIFactory` interface. The reasons behind this apparently complex architectural option are extensively discussed in [8]. Nevertheless, the general idea is simple. Each object (local or remote) is controlled through a graphical user interface that always has to run locally. Therefore, the latter has to be dynamically downloaded from the room server machine to the client avatar machine. To transparently achieve this goal, we use the factory pattern (see [22]) twice. The first factory, `WObject`, generates `UIFactory` objects, which themselves are `WObjectGUI` objects factories. This approach was inspired by one of the first successes of the Jini.org Jini Community process, the ServiceUI project, led by Bill Venners of Artima Software [23].

6. ACHIEVEMENTS

Considering the actual version of the MaDViWorld prototype, its main benefits are commented upon in a few words as follows.

- MaDViWorld is written in *pure Java* and benefits from all the related advantages. Particularly, it is platform independent and every computer with the standard JRE installed can use it.
- An *original and powerful use of Java RMI* and of its basic facilities, essentially object serialization and dynamic class loading, provides the cornerstone of MaDViWorld.
- MaDViWorld is for a *large audience*. There are no other material requirements besides a standard personal computer with a connection to a local network or the Internet.
- Virtual worlds built over the MaDViWorld framework are *really distributed*. Their rooms are spread over several computers connected via a network, and there is no need for a single host to have full knowledge of the world.

[†]Integrated Development Environment.

Table I. Comparison between MaDViWorld and some of the main Java technologies.

	MaDViWorld	Applets	Java Web Start	RMI	JINI	JavaSpaces
Methods[a] can be executed locally	×	×	×		×	×
Methods[a] can be executed remotely	×			×	×	
Persistence	×				×	×
GUI of service/objects	×	×	×			
Class mobility[b]	×	×	×			
Object mobility (from host to host)	×			×[c]	×[c]	×[c]

[a]Of the objects/services provided.
[b]Class mobility is not just dynamic class loading, but the physical downloading of class files.
[c]Only under the condition that the HTTP server initially referenced by the object's codebase is up and running. This is not necessary in MaDViWorld.

- MaDViWorld offers a very effective *persistence* mechanism. Each server guarantees the persistence of the part of the world it manages.
- Our framework is *extensible*. By merely respecting some standard interfaces, any Java programmer can customize and extend MaDViWorld or create her own classes of objects. Then, the end-user can add these completely new objects to the world while the system is running.
- MaDViWorld offers *class and object mobility* facilities. Objects are installed and moved around transparently, without the user or even the vi-world developer having to worry about the peculiarities of these tasks. Within MaDViWorld the avatar client applications automatically download all the code needed to interact with the objects selected by its user. This is a vital requirement for a flexible and extensible distributed virtual world, since the client application does not know *a priori* the particular implementation of the objects it will handle.
- All avatars in a common room *share the same objects* hosted on the server, which has the sole responsibility for managing their states and persistence. So the clients do not need to locally cache a part of the world and it is the task of the server to manage events distribution.
- Under the virtual world metaphor, MaDViWorld tackles many interesting and very technological issues. MaDViWorld is at *the crossroad of many Java products*: applets, Java Web Start, RMI, Jini and Javaspaces. It has characteristics of each of these and thus represents an original combination of their features. Table I shows a comparison of MaDViWorld and these other technologies.

Finally, note that the goal of MaDViWorld was to implement a really distributed virtual world only using Java remote method invocation distributed technology and some basic Java features, and our prototype shows that this is feasible in many interesting ways. So we clearly decided that *MaDViWorld would not be a new Java middleware system*, like FlexiNet [24] and we chose not to use a legacy middleware platform that supports mobile services like Voyager [25]. We also ruled out working with one of the many 'agent' systems available, because the majority of these systems basically consist of mobile script platforms that allow scripted agents to move around in a distributed network.

Examples of these systems include Aglets [26]. Nevertheless, in order to have all the pieces of this 'puzzle' fit together, we needed some 'glue'. Object providers and their users (rooms and avatars) have to agree upon a general collection of interfaces defining a standard protocol. The clearly defined and well-separated MaDViWorld specification level with its interface files shared between avatars, wizards and servers provides us this 'glue'.

To conclude this chapter, let us mention the official website of the MaDViWorld project (http://diuf.unifr.ch/sde/projects/madviworld/). This site provides the following:

- the whole source code of the latest MaDViWorld version, which can be freely downloaded;
- an installation guide;
- the up-to-date javadoc documentation of the framework;
- actual publications and links to on-going projects related to MaDViWorld.

7. FUTURE WORK

The current version of MaDViWorld is a functional one and is currently being extensively used by some of our students within their yearly computer science projects (see Section 4.3). It is, however, the framework's first stable version. Therefore, many improvement efforts are still needed. These improvements will clearly be divided into two directions: (1) a revision of the specification protocol under the condition of remaining fully backward compatible; and (2) improving the concrete implementation.

7.1. Specification aspects

The specification protocol of MaDViWorld has to be reviewed in more detail. The methods that have to be offered in the interfaces of the specification layer need to be refined. Some of the aspects that should be considered but are still not present in the current version of MaDViWorld are listed below.

- The `RoomFactory` should be much closer to a *room manager*. A `removeRoom()` method, for instance, is required.
- The *security model of the world*, i.e. of the rooms, is not well-defined. The actual prototype just provides a minimal *ad hoc* security model. In a more elaborated version, rooms should have a password associated with them and thus the new methods `setPassword()` and `changePassword()` should be available so the room would be able to give access to methods like `changePermission()`, `changeKind()`, `changeOwner()`, `changeTopology()` only to the owner.
- The object concept is still too poor and should be extended. *Objects should interact* with each other: changes in a given object should be allowed to induce changes in other objects of the room. For instance, if we have a box and we put objects into this box, then moving the box implies that we also move all the objects in the box. This concept implies that some *objects are containers* of other objects (e.g. boxes, tables, bookshelves, etc.) and naturally, these new objects induce an *object visibility concept*. There would be closed (a box) and open (a box, a table) objects and their visualization would be treated differently. The interested reader can find in [14] an extensive discussion of such a model.

- Avatars, objects, rooms and their doors should have concrete *topological and geometrical attributes* like `size`, `color`, `coordinates`, etc. Constraints induced by such information should be managed by the world. So the definition of 'moving' will change. In the actual version avatars and objects move only when they change container.

7.2. Implementation aspects

On the implementation side there are still many improvements to be made. Some of them are the direct consequence with respect to the additional specifications we have just mentioned. Below, we give a non-exhaustive list of what these challenges may be.

- MaDViWorld uses the *java security model* and its security policy files. In the future, we will have to provide specific policy files adapted for room servers, avatars and so on.
- Actually, firewalls may prevent the use of our framework on the Internet global network, but the application of the widely used *HTTP tunneling* method (see [27]) can be used to solve this problem.
- The Jini Discovery and Lookup services should be used as a naming service for RMI in order to *replace the classic rmiregistry*, which has two major limitations: its 'flat' name space and lack of persistence. First, this solution would allow avatars to look up rooms by other means than their names, namely by a matching mechanism on service types and attributes (see [28]). The user could then easily find all the 'arcade game rooms' or 'spanish chat rooms' for instance. Secondly, it would allow a looser relation between the rooms and the server hosting them. One will then be able to create a room on a first machine of the LAN/WAN and, after a crash, relaunch it on another machine.
- Some implementation details should be optimized. For instance, we use the serialization classes of `java.io` to store the states of the rooms, of the avatars and of the room factory into files. It would be conceivable to use the *compression classes* of `java.util.zip` to keep the size of these files as small as possible. Furthermore, the MaDViWorld packages, as well as the object packages, could be stored into *JAR files* to improve class downloading.
- The `getInformation()` method of the `WContainer` interface should not just return a short description string. An *XML* file containing all the *topological, geometrical* and other relevant *information* would perhaps be the best alternative. The client applications should then show other avatars, objects and the rooms in a more *sophisticated graphical model* (2D or 3D) and the rooms should provide an adapted events broadcasting mechanism.
- For communications implying real-time audio or video, a *streaming mechanism* (UDP, RTP) should be provided. An appropriate strategy within the MaDViWorld framework could be the following. First we provide a set of standard methods allowing the vi-world objects to open communication channels. The `WObjectImpl` class (see Figure 7) is clearly a first choice candidate where these methods could be implemented. Secondly we extend the avatar's capabilities in such a way that it can be told to connect to such channels. This extension implies adding the appropriate methods to the `Avatar` remote interface.
- Using Java ORB over RMI could give the potential to *access objects written in other languages* (e.g. C++) and to exploit CORBA services (see [6]).

ACKNOWLEDGEMENTS

We would like to thank the anonymous referees for their detailed and insightful comments on earlier versions of this paper.

REFERENCES

1. Microsoft Corporation. *Virtual Worlds Group*. http://www.vworlds.org [4 February 2002].
2. Eames R. *The Future of the Virtual Worlds Platform*. http://vworlds.research.microsoft.com/Docs/Future/VWFuture.htm [1999].
3. Fabre Y, Pitel G, Soubrevilla L, Marchand E, Géraud T, Demaille A. A framework to dynamically manage distributed virtual environments. *Virtual Worlds, Proceedings of the Second International Conference, VW 2000*, Paris, France, 5–7 July. Springer: Berlin, 2000; 54–64.
4. Frécon E, Stenius M. DIVE : A scaleable network architecture for distributed virtual environments (special issue on Distributed Virtual Environments). *Distributed Systems Engineering Journal* 1998; **5**(3):91–100.
5. Greenhalgh C, Benford S. MASSIVE: A distributed virtual reality system incorporating spatial trading. *Proceedings 15th International Conference on Distributed Computing Systems*. IEEE Computer Society Press: Vancouver, Canada, 1995, 27–34.
6. Wilson S, Sayers H, McNeill MDJ. Using CORBA middleware to support the development of distributed virtual environment applications. *Proceedings of the 9th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2001 (WSCG'2001)*, Plzen, Czech Republic, February 5–9, 2001.
7. Deriggi FV, Kubo MM, Sementille AC, Brega JR, dos Santos SG, Kirner C. CORBA platform as support for distributed virtual environments. *Proceedings IEEE Virtual Reality*. IEEE Computer Society Press: Los Alamitos, CA, 1999, 8–13.
8. Fuhrer P, Kouadri Mostéfaoui G, Pasquier-Rocha J. 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, July 2001.
9. Diehl S. *Distributed Virtual Worlds: Foundations and Implementation Techniques Using VRML, Java and CORBA*. Springer: New York, 2001.
10. Kirsh D. Adaptive rooms, virtual collaboration, and cognitive workflow. *Cooperative Buildings—Integration Information, Organization, and Architecture* (*Lecture Notes in Computer Science*), Streitz N (eds.). Springer: Heidelberg, 1998; 94–106.
11. Reid E. Cultural formations in text-based virtual realities. *Masters Thesis*, English Department, University of Melbourne, 1994.
12. *The Mud Connector*. http://www.mudconnector.com [4 February 2002].
13. Pavel C. Mudding: Social phenomena in text-based virtual realities. *Proceedings of the 1992 Conference on Directions and Implications of Advanced Computing*, Berkeley, May 1992; 26–34.
14. Vellon M, Marple K, Mitchell D, Drucker S. The architecture of a distributed virtual worlds system. *Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, Santa Fe, New Mexico, April 27–30. Usenix Association: Berkeley, CA, 1998.
15. Macedonia MR, Zyda MJ. A taxonomy for networked virtual environments. *IEEE Multimedia* 1997; **4**(1):48–56.
16. Papert S. *Mindstorms: Children, Computers, and Powerful Ideas*. Harpercollins: New York, 1999.
17. Abelson H, Disessa AA. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. MIT Press: Cambridge, MA, 1986.
18. Flanagan D. *Java Examples in a Nutshell* (2nd edn). O'Reilly and Associates: Sebastapol, CA, 2000.
19. Flanagan D, Farley J, Crawford W, Magnusson K. *Java Enterprise in a Nutshell*. O'Reilly and Associates: Sebastapol, CA, 1999.
20. Li S. *Professional Jini*. Wrox Press Ltd.: Birmingham, 2000.
21. Oaks S, Wong H. *Jini in a Nutshell*. O'Reilly and Associates: Sebastapol, CA, 2000.
22. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series: Reading, MA, 1995.
23. Jini ServiceUI Project. http://developer.jini.org/exchange/projects/serviceui/ [4 February 2002].
24. Herbert AJ, Hayton RJ, Bursell M. Mobile Java objects. *BT Technology Journal* 1999; **17**(2). *Networked distributed systems*, http://www.bt.com/bttj/.
25. Recursion Software. *Recursion Software: Products – Voyager 4.5*. http://www.objectspace.com/products/voyager/ [4 February 2002].
26. IBM. *IBM Aglets Software Development Kit*. http://www.trl.ibm.com/aglets/ [4 February 2002].
27. Grosso W. *Java RMI*. O'Reilly and Associates: Sebastapol, CA, 2002.
28. Pitt E, McNiff K. *java.rmi : The Remote Invocation Guide*. Addison-Wesley: Harlow, 2001.

