MaDViWorld Objects
Examples and Classification

Patrik Fuhrer and Jacques Pasquier-Rocha

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

Abstract. The aim of this paper is to present some concrete possibilities of the MaDViWorld framework. It contains several examples of implemented "active objects" and proposes a first attempt at classifying them. This paper also provides a comparison with agent technology and positions our virtual world objects in relation to mobile agents.

Keywords: Software Framework, Virtual World, Agents

Contents

1 Introduction

The Software Engineering Group at the DIUF[1] has developed an object oriented distributed framework supporting massively distributed virtual worlds, called MaDViWorld. The general concepts have been introduced in [7] and an insight of the global framework architecture is provided by [6]. Some more specific aspects such as the distributed event model or the object structure are documented in [10] and [8]. A more theoretical paper [9] focused on the underlying formal model for distributed virtual worlds.
This paper gives an insight of the possibilities of the MaDViWorld framework with the help of a typical scenario and the presentation of some existing objects. Section 4 then discusses the relationship between agents and the virtual world objects. Finally, Section 5 provides a short discussion about the code mobility issue.

2 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 in this room.
- Figure 1b): Sylvia and Hans both launch the battleship game and start playing it (cf. Figure 2).
- 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 put the game she had in her bag into the room. Then both Hans and Sylvia launch the game again 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 reveal several interesting points:

- The remote event mechanism plays an important role at two levels in the 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 "tic tac toe" object in room R2 (cf. Figure 3). 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 (cf. Figure 2).
- The battleship object has "physically" been carried from room R1 to room R2 by the avatar Sylvia. It is worth noticing 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.

3 Some Objects

It is rather simple[2] for a Java programmer using the framework to develop his own objects. The object developer is free to decide if her new object will:

[1] Department of Informatics of the Univesity of Fribourg, Switzerland

[2] The interested reader is referred to the MaDViWorld's official web site [5], where she can both download the code and consult an HTML step by step cookbook guide to programming MaDViWorld objects [13].

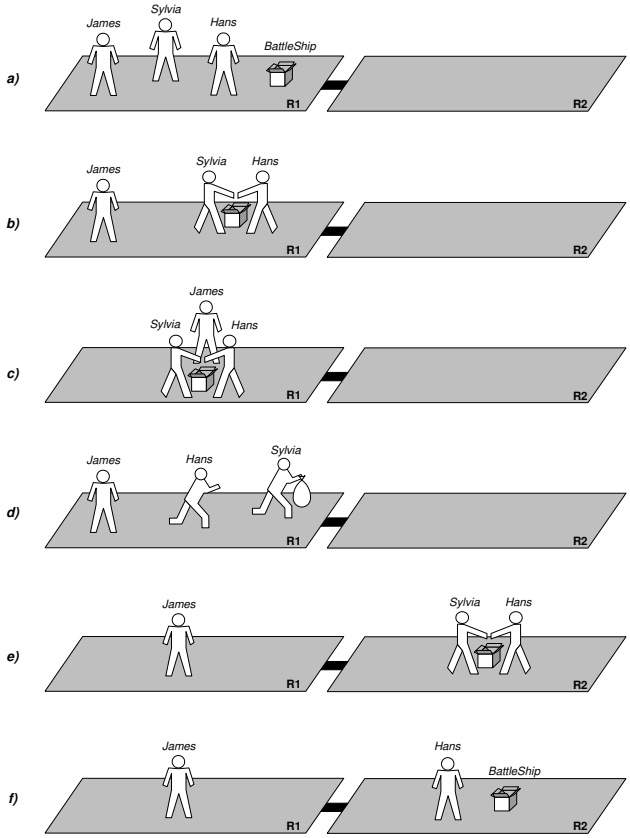

Figure 1: A typical scenario in MaDViWorld.

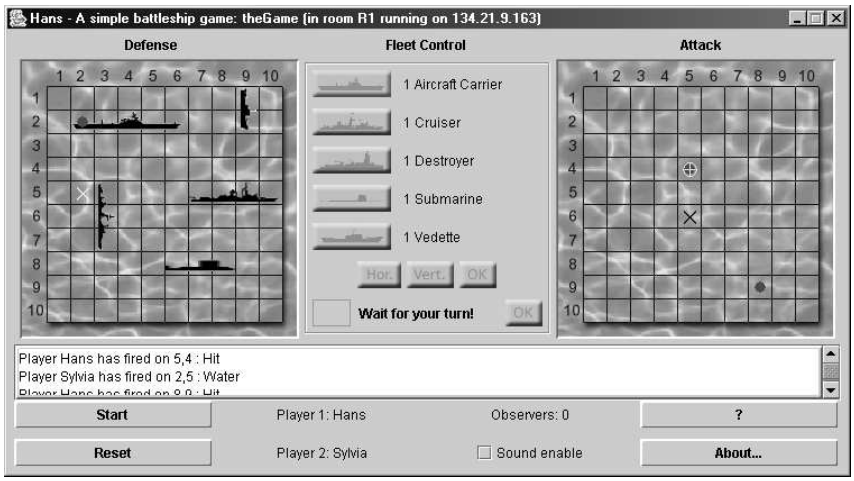

Figure 2: A battleship game.

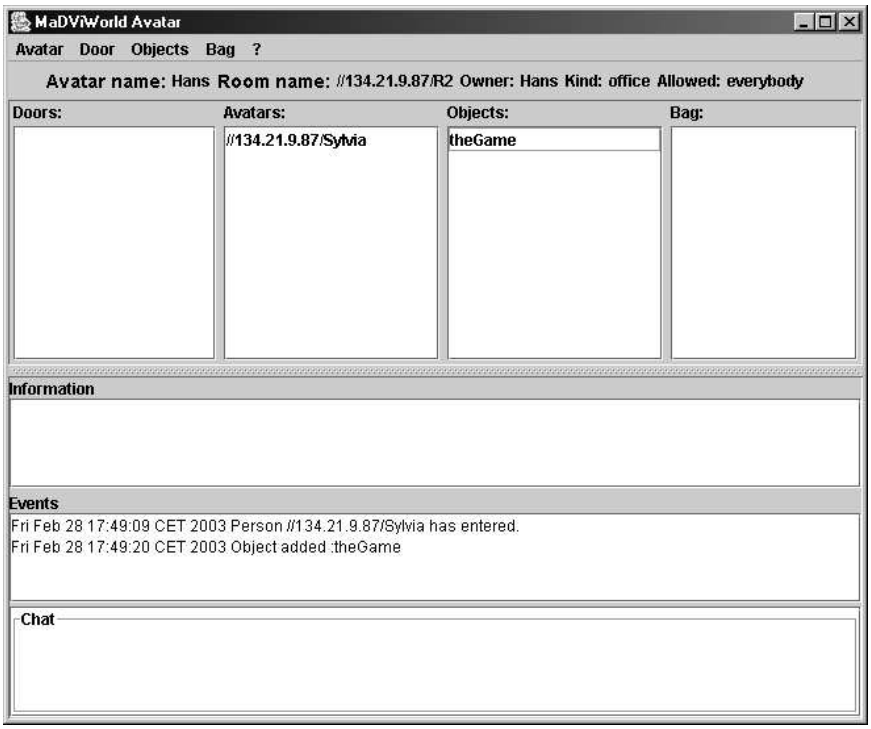

Figure 3: A MaDViWorld "basic" avatar.

- be statefull or stateless, determining if the internal state of the object is carried around when the objects moves or not.
- be single-user or multi-user;
- take advantage of the distributed event mechanism to have a reactive GUI (intra-communication) or to communicate with the other objects located in the same room by multicasting events (inter-communication).

Some of the already existing objects help illustrating these different possibilities and are classified below into three categories.

3.1 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 *resource sharing*. Objects needing a lot of computing power are put in a room hosted by a powerful computer and remotely started and driven by thin avatar clients running the object's GUI. A little example illustrating this feature is the fibonacci number calculator. A typical example object illustrating the fact that the object is using the resources of the distant machine hosting the room in which the object is contained in, is the clock. Indeed, if an avatar launches a MaDViWorld clock object, she will see the time of the remote machine and not the time of the machine hosting the avatar and running the clock's graphical interface. This example also illustrates well that one single object implementation can have several graphical user interfaces (cf. Figure 4).
A lot of other examples can easily been imagined, for example from mathematical topics such as numerical linear algebra, fractal calculation, cryptography or linear programming solvers.

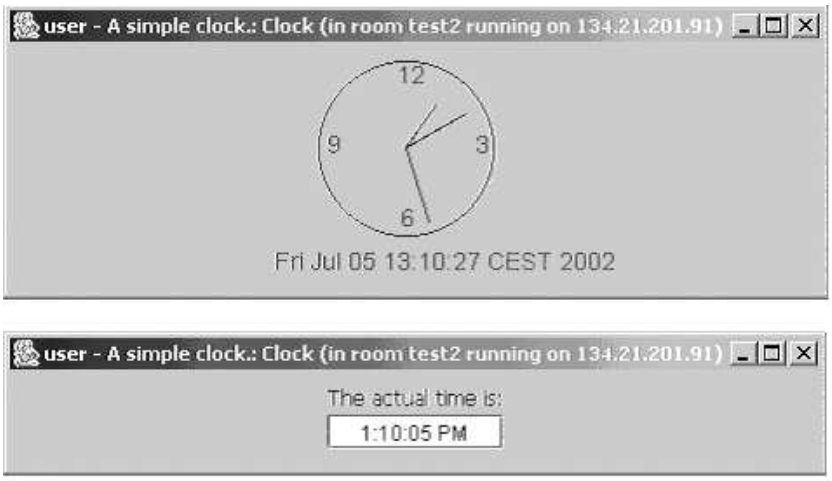

Figure 4: A clock object in MaDViWorld (analogical and digital GUI).

3.2 Social Objects

The remote event mechanism model can also be used in order ot make objects communicate with each other. A possible application consists in creating so called *"social" objects*. For example, one can create a virtual pets community. 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 5. Other applications of the communication between objects are, for instance, an audio player accessing a music rack containing several music files. Another example are robots fighting against each other on a play ground.

3.3 Collaborative Objects

The framework offers all what is needed in order to build *collaborative objects*. Indeed, objects are automatically shared by several users and the events are transparently broadcasted. This allows the creation of collaborative editors or "chat" utilities. Multi-player games are also part of this category of objects. Existing examples of multi-user games are the battleship game (cf. Figure 2) or the "tic-tac-toe" game (not shown). The minesweeper game shown in Figure 6 is the typical example of a single-user game. It ranges in the collaborative objects category if one considers the avatars watching how someone else plays.

4 Agent Technology

The goal of this section is to position our work with respect to the emerging research field of mobile software agents. Indeed, the reader could ask himself if the the objects in our distributed virtual worlds are not agents.

4.1 What Is An Agent?

The word agent has found its way into a number of technologies. It has also been applied early to constructs, which were developed for improving the experience provided by

Figure 5: A virtual pet in MaDViWorld.

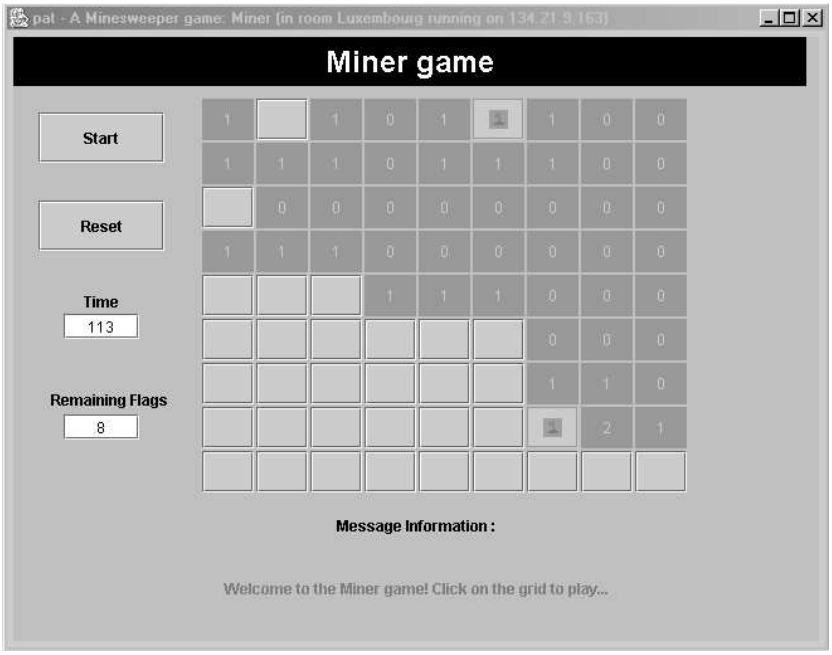

Figure 6: A minesweeper game object in MaDViWorld.

collaborative online social environments like MUDs and MOOs.
There is a plethora of definitions for software agents (cf. for instance [4], [16], [12] or [14]) and there is no consensus on a generally accepted precise definition. A thorough, well-thought-out classification scheme is given by [4], where one can also find a very broad definition[3], which is unfortunately too large to be useful as is. But even if there is no general agreement as to what constitutes an agent, or as how agents differ from programs, we can provide a list of characteristics that have been proposed as desirable agent qualities (cf. [1], [4], [17] and [2]):

- reactive: responds in a timely fashion to changes in the environment. In other words, an agent senses its environment and acts upon it,
- autonomous: is able to take initiatives and exercices a non-trivial degree of control over its own actions,
- goal-oriented: does not simply act in response to the environment (pro-active),
- temporal continuity: is a continually running process, not a "one-shot" computation that maps a single input to a single output, then terminates,
- communicative: communicates with other agents, perhaps including people, in order to obtain information or to enlist their help in accomplishing its goals (socially able),
- collaborative and/or competitive: cooperates and/or competes with other agents in pursuit of common goals,
- learning: changes its behavior based on its previous experience and automatically adapts to changes in its environment (adaptive),
- mobile: able to transport itself from one machine to another (possibly across different system architectures and platforms),
- flexible: actions are not scripted; it is able to dynamically choose which actions to invoke, and in what sequence in response to the state of its external environment,
- character: a well-defined believable "personality" and emotional state.

Every agent, by the definition of [4] satisfies the first four properties. Adding other properties produces potentially useful classes of agents, for example, mobile, learning agents.
Of course, other classifying schemes are possible. For example, software agents are often classified according to the tasks they perform and following kinds can be identified: *interface agents*, supporting the metaphor or a personal assistant, which is collaborating with the user and which employs artificial intelligence (cf. [11]); *information agents*, which have access to at least one information source and are able to collate and manipulate information obtained from these sources to answer queries posed by users and other information agents (cf. [17]); *entertainment agents* like Julia (cf. [3]) integrating reactivity, goals and emotion; *commerce agents* providing commercial services (e.g. selling, buying, prices' advice) for a human user or another agent; or even *computer viruses*.
Finally, from the preceding considerations it becomes clear that (software) agents do not exist by themselves but that they "live" in an environment, that we will call an *agent host*. The basic requirements of an agent host, are identified in [15]:

- An agent host must allow multiple agents to co-exist and execute simultaneously.
- An agent host must allow agents to communicate with each other and itself.
- It must be able to negotiate the exchange of agents.
- It must be able to freeze an executing agent and transfer it to another host.
- It must be able to thaw an agent transferred from another agent host and allow it to resume execution.
- It must prevent agents from directly interfering with each other.

[3] An *autonomous agent* is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

4.2 Are The Virtual World Objects Agents?

As seen in Subsection 3, the MaDViWorld framework allows for the creation of objects of several types. According to the criteria discussed in Paragraph 4.1, it is also possible to create software agents.
As a matter of fact, a framework user could develop an object that is reactive, autonomous, temporally continuous, communicative and mobile by directly and effortlessly using the offered features. The object developer is then free to add goal-orientation, flexibility or learning capabilities to her object in order to build a more interesting software agent.
Among the already existing objects, the virtual pet example (mentioned in Subsection 3.2) could be classified as a basic agent. Indeed:

- It is reactive, as it senses the death of another pet and is affected by this event.
- It is autonomous, as it starts "living" automatically.
- It is goal-oriented, with the most basic goal of just "living".
- It is mobile, as an avatar can move it from one room to another.
- It is temporally continuous, as if it moves from one room to another, it continues to live at the point it was before moving.
- It is communicative, as it can inform other pets that it is dying.

The MaDViWorld platform offers several features that have to be supported by an agent architecture. However, the cognitive characteristics, such as adaptation, learning and goal orientation are not present in our software solution. And it seems that these are the areas (the aspect related to intelligence) that receive the most attention in the agent community. To conclude, one could say that the main difference is the intent of the MaDViWorld objects. At the beginning, they were not developed to act like agents, but the framework happens (it is a side effect) to potentially support these kind of objects as well!

5 Code Mobility

Code mobility is an important technical issue for the development of mobile objects. For distributed virtual worlds, it is important to have a framework supporting strong code mobility.
The classical Java code mobility is achieved by dynamic classloading. The bytecode of a new object is remotely downloaded from an HTTP server thanks to a codebase annotation. This elegant solution has the drawback that during the whole object lifetime, the codebase annotation remains the same and always points to the original HTTP server.
In the context of massive distributed virtual worlds, where objects move around frequently, this becomes a problem when the original HTTP server crashes or simple does not exist any more or is connected with a slow network: the object can not move anymore and the HTTP server becomes a single point of failure.
What a distributed framework should provide, is a total transitivity of the object bytecode. The codebase annotation should change during the object lifetime, always pointing to the latest HTTP server. This concept is provided by MaDViWorld and is referred to as strong code mobility (cf. Figure 7).

6 Conclusion

The aim of this paper was to illustrate the potential use of MaDViWorld "active objects". Several student projects are concerned with the study and the development of such objects. These projects provided a serious test bed for the development and validation of the framework. We are thankful for all of them. The interested reader can consult [5] for an actual description of the completed and ongoing projects.

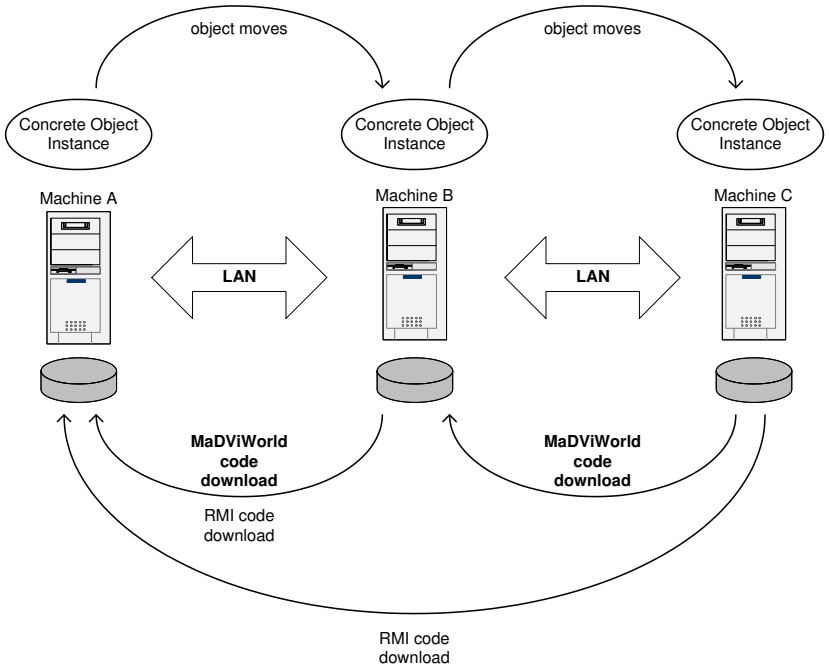

Figure 7: Strong code mobility in MaDViWorld.

References

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

[2] O. Etzioni and D. Wedd. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–79, 1994.

[3] L. N. Foner. Entertaining agents: A sociological case study. In *Proceedings of the First International Conference on Autonomous Agents (AA '97)*, 1997.

[4] 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.

[5] P. Fuhrer. MaDViWorld (Massively Distributed Virtual Worlds). on-line, accessed on 21st August 2003. http://diuf.unifr.ch/softeng/projects/madviworld/.

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

[7] 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.

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

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

[10] 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.

[11] P. Maes. Social interface agents: Acquiring competence by learning from users and other agents. In *Proceedings of the 1994 AAAI Spring Symposium on Software Agents*, pages 71–78. AAAI Press, March 1994.

[12] P. Maes. Artificial life meets entertainment: Interacting with lifelike autonomous agents. *Communications of the ACM, Special Issue on New Horizons of Commercial and Industrial AI*, 38(11):108–114, November 1995.

[13] F. Marchon and P. Fuhrer. MaDViWorld object porgrammer's guide. on-line, accessed on 21st August 2003. http://diuf.unifr.ch/softeng/projects/madviworld/guide/.

[14] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice-Hall, 2nd edition, December 2002.

[15] T. Sundsted. An introduction to agents, find out what agents are and what they can do for us, and take the first steps toward building your own simple agent architecutre in java. *JavaWorld How-To-Java*, June 1998. http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html.

[16] M. Wooldridge. Agent-based software engineering. *IEEE Proceedings on Software Engineering*, 144(1):26–37, 1997.

[17] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.