Munchkins: A Shell for Distributed Multi-User Games

Martin Henz, Martin Müller, Markus Wolf

Abstract— **Multi-user dungeons (MUDs) are text-based computer games being played over the Internet. They are usually based on a simple client-server architecture, allowing for their wide availability, but also imposing serious limitations upon their users. In this paper, we propose an implementation of a MUD as a truly distributed application using the concurrent object-oriented programming language Oz to overcome these limitations.**

Keywords— **Oz, multi-user dungeons, distributed programming**

1. Introduction

Distributed programming is gaining importance in today's globally networked computing. Concurrent objects allow real-world concurrency of distributed applications to be naturally modeled. Our research concentrates on the concurrent object-oriented programming language Oz [8]. Our aim is to explore the possibilities and limitations of the language for distributed applications through practical applications.

In this paper, we focus on distributed multi-user games (MUDs). The term MUD originally stood for *Multi User Dungeon*, today the term *Multi User Dimension* is preferred. They have a long history as games allowing several players to interact with each other in a text based virtual reality, [1]. Recently, more serious applications in the context of cooperativity [5], [4] or education [3] have been considered.

Conventional MUD systems consist of one server to which the user connects via a client (often telnet). The code for all the objects that participate in the virtual environment resides on the server. Often the single server can not adequately handle the load of large amounts of people using it. The concept of virtual rooms with defined exits suggests distributing the rooms (or collections of rooms) to several servers. This lessens the workload of the individual servers and eases coding.

We chose distributed MUDs for studying distributed programming in Oz for the following reasons:

- The advantages and problems of distribution can be identified by comparing our distributed MUD with its existing non-distributed cousins.
- MUDs clearly show some of the requirements that languages for distributed computation must fulfill such as fault tolerance, network transparency, persistence and environments for distributed programming.
- By allowing Oz programmers around the world to extend a running MUD, experience in large scale distributed applications can be gained.
- MUDs are lots of fun...

To achieve reusability, we implemented a shell for designing MUDs. We call the shell *Munchkins*[1]. Actual MUDs can then be obtained by writing a game specific library.

Increasing commercial interest in online services lead to the reuse of MUD-related technology for commercial products that provide virtual worlds to their customers.

Most of these applications, however, inherit the limitations of a client-server architecture, and are far less extensible than traditional MUDs.

2. Current Implementation

We implemented *Munchkins*, a shell for designing distributed MUDs. A *Munchkins* game allows players to enter a virtual world of rooms, other players and other objects and to interact with them. The virtual world is organized in sites, each hosting a number of objects. *Munchkins* players can dynamically add new objects to sites and new sites to the game.

Local and remote objects are currently not completely interchangable from the user's perspective. Completely hiding this distinction would be desirable to provide the illusion of a global object space. Network transparency is currently violated in a number of other cases and needs further work.

We are currently building a MUD on top of *Munchkins*, that provides access to a virtual version of the building in which the DFKI Programming Systems Lab is located. Players will be able to visit (virtual versions of) the rooms of the researchers and contact them through the game.

[1]Inhabitants of the land of Oz in [2]

In a first step, we concentrate on the aspects of distribution and therefore use a simple hyper-text based user interface. We are aware of the fact that only a minority of computer users will be attracted to such a gaming environment. In particular, we hope to attract a community of Oz programmers in academia as game players and wizards to keep *Munchkins* alive and thriving.

3. Discussion

We identified the following advantages particular to Oz for developing *Munchkins*:

Concurrent Objects [6] are directly available and can be mapped to the objects present in the game.

Constraint-based Communication proves to be superior to message passing, allowing techniques such as incomplete messages and special purpose communication protocols.

Interoperability features of Oz [7] allow easy programming of network communication on the level of internet sockets.

The following problems and deficiencies of Oz have been identified during the development of *Munchkins*.

Persistence of Distributed Objects. In current Oz, it is difficult to implement persistent objects. Thus, recovery from network and site failures is currently not possible.

Network Transparency. Transparent representation of remote and local data is difficult to achieve and currently only approximated.

Fault Tolerance. A system that is dynamically extended by programmers around the world has to be robust against programming errors. Programming concepts are needed for fault tolerant distributed programming.

Distributed Programming Environment. The lack of a distributed programming environment currently forces the programmers of *Munchkins* to communicate source code outside of the application, limiting programming efficiency and security.

In current Oz, we can work on the symptoms of these insufficiencies on a low level. However, to support distributed programming in a satisfactory way, a language design is necessary that incorporates distribution right from the start. Such a language is being investigated in the Programming Systems Lab at DFKI. We hope that experimentation with *Munchkins* will clarify both the advantages and deficiencies of current Oz for distributed programming and provide a platform of experimentation during the design of a distributed programming language on the base of Oz.

References

[1] R. Bartle. Interactive Multi–User Computer Games, MUSE Ltd., UK, Research Report, commissioned by Britisch Telecom plc. ftp://parcftp.xerox.com/pub/MOO/papers/mud-report.ps, 1990.

[2] L. F. Baum. *The Wonderful Whizard of Oz.* G. M. Hill, 1900.

[3] A. Bruckmann. Serious Uses of MUDs? Panel at DIAC 94, Cambridge, MA, April 23-24, 1994. ftp://media.mit.edu/pub/asb/papers/serious-diac94.txt, 1994.

[4] D. V. Buren, P. Curtis, D. Nichols, and M. Brundage. The AstroVR Collaboratory, An On-line Multi-User Environment for Research in Astrophysics. In R. Hanisch, editor, *Astronomical Data Software and Systems IV.* Astronomical Society of the Pacific: San Francisco, 1994.

[5] R. Evard. Collaborative networked communication: Muds as systems tools. In *USENIX '93. Proceedings of the 7th Systems Administration Conference (LISA 93), Monterey, CA, November 1-5, 1993.* USENIX, USENIX, 1993.

[6] M. Henz, G. Smolka, and J. Würtz. Object-oriented concurrent constraint programming in Oz. In V. Saraswat and P. V. Hentenryck, editors, *Principles and Practice of Constraint Programming*, chapter 2, pages 29–48. The MIT Press, Cambridge, MA, 1995.

[7] C. Schulte. Open programming in DFKI Oz. DFKI Oz documentation series, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994.

[8] G. Smolka and R. T. (ed.). DFKI Oz documentation series. Available on paper or via WWW from `http://ps-www.dfki.uni-sb.de/oz/`, Deutsches Forschungszentrum für Künstliche Intelligenz, Stuhlsatzenhausweg 3, D–66123 Saarbrücken, Germany, 1994.