Virtual World Brokerage with BinProlog and Netscape

Paul Tarau
Département d'Informatique
Université de Moncton
E-mail: *tarau@info.umoncton.ca*

Koen De Bosschere
Vakgroep Elektronica
Universiteit Gent
E-mail: *kdb@elis.rug.ac.be*

Abstract

LogiMOO is a BinProlog-based Virtual World running under Netscape 3.0 for distributed group-work over the INTERNET and user-crafted virtual places, virtual objects and agents. LogiMOO is implemented on top of a multi-threaded blackboard-based logic programming system (Multi-BinProlog 5.00) featuring Linda-style coordination. Virtual blackboards allow efficient mirroring and real-time responsiveness over TCP/IP links while Solaris 2.4 threads ensure high-performance local client-server dynamics. Embedding in Netscape allows advanced VRML and HTML frame-based navigation and multi-media support, while LogiMOO handles virtual presence and acts as a very high-level universal object broker.
Keywords: Virtual Worlds, high-level HTML,VRML,CGI programming distributed object brokerage, blackboard-based logic programming, Linda coordination, client-server applications in Prolog, embedded logic engines

1 Introduction

Virtual Worlds are a strong unifying metaphor for various forms of net-walk, net-chat and INTERNET-based virtual presence in general. They start where usual HTML shows its limitations: they do have state and require some form of virtual presence. "Being there" is the first step towards full virtualization of concrete ontologies, from entertainment and games to schools and businesses.

New attempts with various flavors of declarativeness and sophisticated procedural and object abstraction like Microsoft's Active VRML or SGI's Moving Worlds go boldly as close as possible to their strangely special-purpose and insidiously universal objective: representation of (3-dimensional) space combined with time/action. Roughly speaking, Active VRML is functional, Moving Worlds is object-oriented. Both have little to do with logic programming, at least as we know it.

MUDs and MOOs (Multi User Dungeons - Object Oriented) have started with virtual presence and interaction. Again, their links with logic programming are very weak, among the few hundred we know about none seem implemented in a logic programming language.

Some fairly large-scale projects (Intel's Moondo, Sony's Cyber Passage, Black Sun's CyberGate, Netscape's CoolTalk+Live3D, Worlds Inc.'s WorldChat) converge all towards a common interaction metaphor: an avatar represents each participant in a 3D multi-user virtual world. Information exchange reuses our basic intuitions with almost instant *learnability* for free. Their industrial potential is enormous as they are the most natural way to describe virtual organizations/societies. The perfect 'software metaphor' of the next 10 years is here!

What they still miss is quite easy to guess and, arguably, quick to come anyway, with or without logic programming. A key element is programmable intelligent avatars. A second one is a distributed dynamic object model. A third one is unlimited, flexible and secure meta-programming. It seems a few steps beyond the Java+JavaScript combination, but definitely following their objectives if not their way. But the most important one is a unifying model, the *glue* that makes it all as simple to the implementor as it is for the VRML traveler embodied in its avatar.

Our attempt is to make all this obvious through use of some of our favorite logic programming tools, embedded in a state of the art multi-paradigm and multi-platform environment.

We will show that logic programming has the potential to provide this essential "glue" functionality and to cooperate smoothly with specialized tools in a multi-paradigm environment. Centered around the use of LogiMOO [DBPT96], a very high level kernel for INTERNET collaborative work, we will emphasize how easy it is to build distributed applications in a logic programming language featuring Linda-based coordination libraries and how agent programming and live interaction are programmed in a few lines of Prolog. We will show that embedded in a multiparadigm/multi-platform framework through use of standard tools like Netscape 3.0, CGI-programming and independently designed VRML 2.0 based 3-D worlds, LogiMOO closes the gap between logic programming languages and real life INTERNET programming.

1.1 The LogiMOO kernel: a short look from the lobby

The LogiMOO kernel behaves as any other MOO except that interactive Prolog syntax is used. A basic natural language parser allows people unfamiliar with Prolog to get along with the basic activities of the MOO: place and object creation, moving from one place to the other, give/sell virtual objects, talking ('whisper' and 'say'). At login time, a main interactive shell and background notifier (for messages and events targeted to the user) are created. Netscape 3.0 is used to implement a more convenient *remote toplevel with* shorthand com-

mands and the ability to pass along full Prolog queries. Basic visual feed-back is provided on the people and objects in the neighborhood and the ports towards other virtual places. Objects in LogiMOO are represented as hyper-links (URLs) towards their's owners home pages where their 'native' representation actually resides in various formats (HTML, VRML, GIF, JPEG etc.).

2 Blackboard-based logic programming

Before entering into the details of the LogiMOO kernel we will give some 'hints' on the set of tools it is built upon.

LogiMOO is built upon Multi-BinProlog [DBT93a, DBJ93], i.e., BinProlog enriched with a Linda-like tuple space. The Multi-BinProlog blackboard is not just an add-on, but is really integrated in the BinProlog environment, featuring backtracking, guarded evaluation, etc.

The Multi-BinProlog blackboard comes in two flavors [DBT96]: there are local blackboards, primarily used for efficient communication and synchronization between processes running on the same (multi-)processor, and there are virtual blackboards that are primarily used for communication with processes running on other machines. A virtual blackboard is a kind of alias, or link to another (remote) blackboard. Logically, it cannot be distinguished from the remote blackboard itself. At the implementation level, a virtual blackboard does not contain data, but is just a local representation of the remote blackboard. All operations issued on a virtual blackboard are automatically forwarded to the remote blackboard. Furthermore, the remote blackboard can in turn be a virtual blackboard pointing to yet another remote blackboard. The last blackboard in a chain must always be a physical blackboard containing real data (see Figure 1).

Further characteristics of a blackboard are (i) that it is persistent, (ii) that the data are manipulated associatively (i.e., based on their content, rather than on their address), (iii) all accesses are automatically synchronized. These are precisely the requirements for a MOO: the status of the MOO should be stored somewhere, and it should not be volatile; if we are looking for a particular object in a MOO, we will always refer to it with a name, never with its address location; a MOO is multiuser, so it should be synchronized at any time.

The basic operations on the blackboard are: blackboard creation and deletion, the creation of processes (independently running Prolog goals) (`tellp`), putting Prolog terms on the blackboard (`tellt`), removing terms from the blackboard (`gett`) and checking the presence of terms (`readt`). Both the get and the read primitives can be blocking or non-blocking. Furthermore, there are some more advanced primitives like blackboard operations working on lists of terms instead of one term, operations to collect the complete contents of a blackboard, and so on.

A pure Linda-based language is not immediately suitable to support a MOO

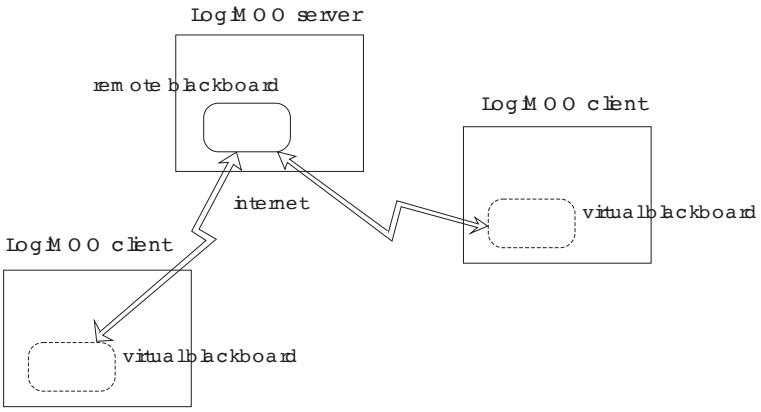

Figure 1: Virtual and remote blackboards

because a Linda-application consists of a number of processes that are created by one application in order to solve a given problem. Here, client processes must be able to connect and disconnect at any time. This requirement can be completely fulfilled by the concept of virtual blackboard. A client wanting to connect with a MOO, creates a local virtual blackboard, hooks it up to the MOO, and from then on, it can interact with the MOO in an unconstrained way by communicating with the local virtual blackboard.

3 The Anatomy of LogiMOO

3.1 The basic LogiMOO to Linda mapping

Clearly representing places with separate blackbords and avatars/objects with processes/terms on local and remote blackboards is the most natural representation, especially when thinking in terms of a distributed "web of MOOs". We will describe the basic ideas behind LogiMoo in terms of a simpler mapping reminescent from client/server architectures, which also allows Netscape based users not having BinProlog on their own computers to access LogiMOO. The "world" is represented by a blackboard on a server, running on a Solaris 2.x system under Multi-BinProlog. Virtual blackboards that "mirror" the "world" on client processes are used if the user has a copy of the Multi-BinProlog and LogiMOO locally. Otherwise, connectionless Netscape 'hits' from the user's computer create short-lived CGI clients on the computer hosting the server[1].

[1] This has been found very useful for teaching applications with low-end PCs.

We will now briefly describe this reduced MOO kernel.[2] *Verbs* available in the MOO are defined through a set of Prolog predicates hiding the complexities of the distributed communication model through the usual metaphors: places (starting from a default lobby), ports, ability to 'move' or teleport from one place to another, a wizard resident on the server, ownership of objects, the ability to transfer ownership and a built-in notifier agent watching for messages as a background thread.

Although Linda operations are definitely lower level than the abstract relationships of time/space ownership, movement, interaction present in a MOO, we have found out that in the context of a logic programming language with unlimited meta-programming abilities like Prolog, representing such relationships is unusually straightforward.

3.2 Basic primitives

Although Multi-BinProlog offers a wide variety of blocking and non-blocking operations, we have tried to express everything in terms of the following basic primitives:

```
% puts X on the server
out(X):- tellt(world,X).

% takes an object matching X from the server
in(X):-gett(world,X),!.

% reads the list Xs matching X currently on the server
all(X,Xs):-bbterms(world,Ts),findall(X,member(X,Ts),Xs).

% starts a background process executing Goal
run(Goal):-tellp(Goal).

% puts private information X on the local default blackboard
local_out(X):-tellt(X).

% checks whether an object matching X is on the local blackboard
local_rd(X):-bbterms(Xs),member(X,Xs),!.
```

This intermediate level of abstraction allows, for instance, single user execution for debugging purposes and an easy porting to another system featuring Linda extensions (like SICStus Prolog 2.x, 3.x), by redefining a small number of basic operations. Resource control is built-in in Multi-BinProlog as blackboards have a user definable size. A blackboard will thus refuse to store information in case it is already full.

[2] Electronically available and remotely executable with Netscape 3.0 from UR `http://clement.info.umoncton.ca/~logimoo` .

3.3 Derived operations

A number of derived operations cover frequently used idioms, some actually replicating existing Multi-BinProlog facilities.

```
% non-blocking read from the server and
% backtrack on objects matching X
rd_all(X):-all(X,Xs),member(X,Xs).

% deterministic read
rd(X):-rd_all(X),!.

% conditional out: puts X on the server unless an object
% matching X is found on the server
cout(X):-rd(X),!.
cout(X):-out(X).

% conditional (non-blocking) in: takes an object
% matching X from the server and fails if no such object is found
cin(X):-rd(X),in(X).

% executes goal G for all objects on the server matching X
forall(X,G):-all(X,Xs),member(X,Xs),G,fail.
forall(_,_).
```

At this time, the resulting extended language is made simpler by defining some of the verbs as prefix or infix operators. A full featured controlled English interface, co-developed with Veronica Dahl and Stephen Rochefort at Simon Fraser University, handling anaphora and coordination, will be integrated in the near future.

3.4 Whoami: managing local information

Non-shared information is kept on the default local blackboard.

The login procedure simply puts the name of the current user on the local blackboard with `local_out(i_am(Name))`, after enforcing unique identity on the server by sending an (encoded) password to the server together with the name. Locally the name chosen by the user is accessible as:

```
whoami(X):-local_rd(i_am(X)).
```

The use of local blackboards allows high-speed access to private state information and allow implementation of sophisticated security protocols.

3.5 Your first servant: the notifier

The notifier is one of the simplest possible *agents*:

```
notifier(Name):-
  repeat,
    in(mes_to(Name,Mes,From)),
    notify(Mes,From),
  Mes==end,
  !,
  halt.
```

The notifier is automatically started from the login predicate as a background thread with `run(notifier(Name))` . No 'busy wait' is involved, as the notifier's thread blocks until `in(mes_to(Name,Mes,From))` succeeds. We refer to [DBT96, DBT93b] for details on the underlying multi-threaded implementation of background goals running under Solaris 2.x.

When `whisper/2`, defined as

```
whisper(To,Mes):-whoami(I),out(mes_to(To,Mes,I)).
```

unblocks the matching `in(mes_to(Name,Mes,From))` , the notifier simply outputs the message with `notify(Mes,From)`.

More complex agents can be programmed at the user level by cloning existing agents and overriding some of their attributes or behavior slots, as far as the underlying Prolog (which is fully accessible from LogiMOO) features an appropriate object system. A quantifier-like notation allows to specify which slots are shared (with bindings flowing unidirectionally from the parent - semantically close to *trait* objects in **SELF** [US87, Smi95]) and which are newly created for local bindings in the clone.

Our work on LogiMOO indicates that a *prototype-based* object-oriented style is particularly appropriate for logic programming languages.[3] BinProlog's object model does not follow the usual class/object/instance made popular by C++ and similar languages. A very simple 'object as sets of assumptions' ontology has been implemented from the start, for reasons of conceptual parsimony. Basically, object composition and class inheritance relations are broken in `part_of` and `clone_of` relations, with appropriate Datalog transitive closure definitions which are incrementally updated to reflect dynamic object creation and/or state mutation. Dynamic and distributed object cloning and 'Java-style' code migration which were already present in the underlying Multi-BinProlog model, allow easy state and source management by humans and software agents.

It is interesting to notice that MOO construction engines naturally generalize currently available object technology. LogiMOO is no exception in this sense.

[3] We have recently found out that this is quite close to SICStus 3.0 object library's delegation-based mechanism, at least in terms of common design objectives.

Distributed abstract 'container' relations materialized as blackboards and the ability to easily program asynchronous agents through a very small set of Linda operations combined with this simple object model ensure a short learning curve and a comfortable level of information sharing and code reuse so that (most of) the programming can be delegated to the users.

3.6 `forall/2`: quantifying over the virtual world

`forall(X,G)` executes a goal G for all objects on the server matching X. It is a convenient way to define a 'broadcasting' verb `yell/1` simply as:

```
yell(Mes):-for_all(online(Name),whisper(Name,Mes)).
```

A less intrusive version `say/1` can be written similarly by restricting it to online users at the current place:

```
say(Mes):-
  whoami(I),
  where(Place),
  for_all(contains(Place,Name),
          (
            rd(online(Name)),
            Name\==I,whisper(Name,Mes)
          )
  ).
```

A Unix-like users predicate listing names of people on the MOO is written as:

```
users:-for_all(user(Name,_,Home),show_user(Name,Home)),nl.
```

3.7 User ontology: verbs and objects

A small number of abstract relationships are enough to describe, a variety of 'surface-level' verbs. `contains/2` is probably the most important one, defined as:

```
contains(Container,Object):-cout(contains(Container,Object)).
```

Non-blocking read `rd/2` can be used to locate a user as in:

```
where(Place):-whoami(Me),rd(contains(Place,Me)).
```

To conditionally create a place (unless it exists) we define:

```
place(Place):-cout(place(Place)).
```

To conditionally create a port (only when the links are already existing places), we define:

```
port(P1,Dir,P2):-rd(place(P1)),rd(place(P2)),cout(port(P1,Dir,P2)).
```

A shorter version with context information will usually be given to the users as:

```
port(Dir,P2):-where(P1),port(P1,Dir,P2).
```

Teleporting is somewhat unrealistic but useful:

```
move(O,P1,P2):-cin(contains(P1,O)),out(contains(P2,O)).
```

On top of teleporting, more realistic movement is implemented as:

```
go(Dir):-
  where(Place),
  rd(port(Place,Dir,NewPlace)),
  whoami(Me),
  move(Me,Place,NewPlace),
  forall(has(Me,O),move(O,Place,NewPlace)).
```

Note that `forall/2` can be used to make someone's belongings follow him. As in the real world, this is usually done selectively on a subset of a user's belongings.

Creating things with `craft/1` marks them with ownership:

```
craft(O):-whoami(Me),
  rd(contains(Place,Me)),
  out(contains(Place,O)),
  out(has(Me,O)).
```

Property transfer (useful for online sales and banking) is prototyped as follows:

```
gives(From,To,O):-
  cin(has(From,O)),
  out(has(To,O)).

give(Who,What):-
   whoami(Me),
   cin(has(Me,What)),
   out(has(Who,What)),
   whisper(Who,'I gave you: '(What)).
```

Although expressing a `sale` predicate in term of give is easy, realistic transactions need a sound security system, not yet implemented in LogiMOO. As usual, PGP-based public-key cryptography can ensure secure transactions on top of an insecure carrier. Although it is possible to implement electronic money and authentication within LogiMOO on top of standard Solaris tools like `des_crypt`, the embedding of LogoMOO in Netscape (described in section 4), allows use of high quality third-party encryption and E-cash technology.

Finally, `look/0` recognizes specific objects and shows them in the most useful form. For instance, under the Netscape interface, users are shown as hyper-links to their home pages and objects belonging to a given user are shown as links relative to the user's home page.

```
look:-
  where(Place),
  nl,writeln(['Ports ',from,Place]),
     for_all(port(Place,Direction,To),
             show_port(Direction,To))
     ),
  nl,writeln(['Online ',in,Place]),
     for_all(online(X),
       (rd(contains(Place,X)),rd(user(X,_,H)),show_user(X,H)
       )
     ),
  nl,writeln(['Sleeping ',in,Place]),
     for_all(user(X,_,H),
       ( rd(contains(Place,X)) ,
         \+ rd(online(X)), show_user(X,H)
       )
     ),
  nl,writeln(['Laying around: ',in,Place]),
     for_all(contains(Place,X),show_object(X)
     ).
```

3.8 Towards "world-wide MOOs"

Instead of one centralized MOO with a number of characters connected to it, one could even think of a fully distributed MOO. In that case, each place could be represented by means of an individual physical blackboard, and moving from one place to another means disconnecting from one MOO-blackboard and connecting to another. The two MOO-blackboards can — but do not have to — be on the same machine. This allows us to create a kind of "world wide MOO" (WWMOO) where users can create their own places, and link them to other places. This is an exciting idea because this allows us to create a really globe-wide MOO where people can just wander around without having to worry

about the precise location of the places. It is really like walking around in a city, browsing around in shops, discovering narrow streets you had never notice before. Furthermore, this world is changing continuously.[4]

A major performance advantage of the distributed MOO is that the load can now be distributed too. On the one hand, the MOO consists of several blackboards possibly running on a set of machines instead of just one, and on the other hand, a user does not connect to the MOO by means of telnet or more expensive GUI-oriented protocols (which puts the complete burden of screen IO on the server machine), but by means of a virtual blackboard. Moreover, local processing of foreign code (similar to Sun's Java language) is possible due to Prolog meta-programming abilities. This means that the often computationally intensive user interface is supported through a local process. The communication between the client and the server can be transparently optimized for speed.

4 LogiMOO as a Netscape 3.0 application

LogiMOO redefines the Prolog toplevel for interacting with Multi-BinProlog either through telnet,/rlogin[5] BinProlog's pipe-based lightweight Tcl/Tk interface, or through the Web with Netscape 3.0. The Web interface uses extensively the latest Netscape goodies:

- Netscape 3.0 frames and forms
- Live3D or CyberGate plug-in for VRML navigation
- JavaScript to help BinProlog control Netscape frames 'from within'
- BinProlog-based lightweight CGI-scripts

Netscape-based users are recognized as special as they do not keep local state (except for their name password and home page kept in the form itself). The stateless CGI Multi-BinProlog client spawn by a "submit" Netscape action connects to a local persistent LogiMOO server. Compilation to C and shared dynamically linked libraries make using BinProlog competitive with Perl scripting.

[4] These features are present to some extent in existing MOOs. For example, one can go to a room that broadcasts all the messages witnessed in another room on another MOO. These facilities are built on top of user-level MOO language and are not well integrated with the MOO software. Our proposed blackboard architecture allows this by design.

[5] Users without their own browser can access LogiMOO through a 'Prolog shell' hosted on our computers with a Unix-level guest account. Setting this up is quite easy by replacing /bin/sh in /etc/passwd with a Multi-BinProlog C-ified executable /opt/bin/mbp, customized to support LogiMOO.

The server periodically saves its state to a file, through a separate background thread while staying responsive to users. A telnet/rlogin-based "wizard" console helps monitoring LogiMOO from a remote computer.

The figure 2 shows the embedding of LogiMOO as a frame-based Netscape application.

Queries are submitted through a CGI POST method. BinProlog reads the standard output using the `CONTENT_LENGTH` environment variable and after a small filter cleans up hexadecimal escapes, it extracts the actual query and its variables through a list-of-characters-to-term conversion. Finally, using a simple trick, we map HTML query syntax to Prolog without any application specific parsing:

```
:-op(1199,xfy,(&)).

(login=L & passwd=P & home=H & query=Query) :-
  login(L,P,H),sit,
  metacall(Query).
```

Objects 'crafted' by users are shown as URL's, relative to their homes. This allows users to 'put' into LogiMOO objects of various formats (VRML, JPEG, WAV, AU) and gives multi-media capabilities for free. LogiMOO keeps the link while the actual object resides on the user's computer accessible by clicking on a link. This makes the link 'hot' so that the user is free to update the actual object without having to notify LogiMOO.

We can even give the illusion that BinProlog commands from within LogiMOO actually allow arbitrary Web navigation through use of a JavaScript 'one-liner', dynamically generated as answer to a query:

```
auto_show(URL,File):-
  make_cmd(['<body
     onLoad="window.open(''',URL,'/',File,''','',_self'')">'],Cmd),
  writeln([Cmd]).
```

For instance, typing:

```
  auto_show('http://eve.info.umoncton.ca:8080/~logimoo','lobby.wrl')
```

in the LogiMOO Prolog query text area, will instantly show LogiMOO's VRML lobby in the Netscape 3.0 output frame, from where the user can further explore links independently. Return to LogiMOO is achieved simply by by clicking on the 'VRML' floor of the room. By using the `_parent` Netscape pseudo-target instead of `_self` the full window is replaced. With `_blank` or a named new target an additional Netscape browser is spawned, allowing independent navigation, while keeping LogiMOO on-screen.

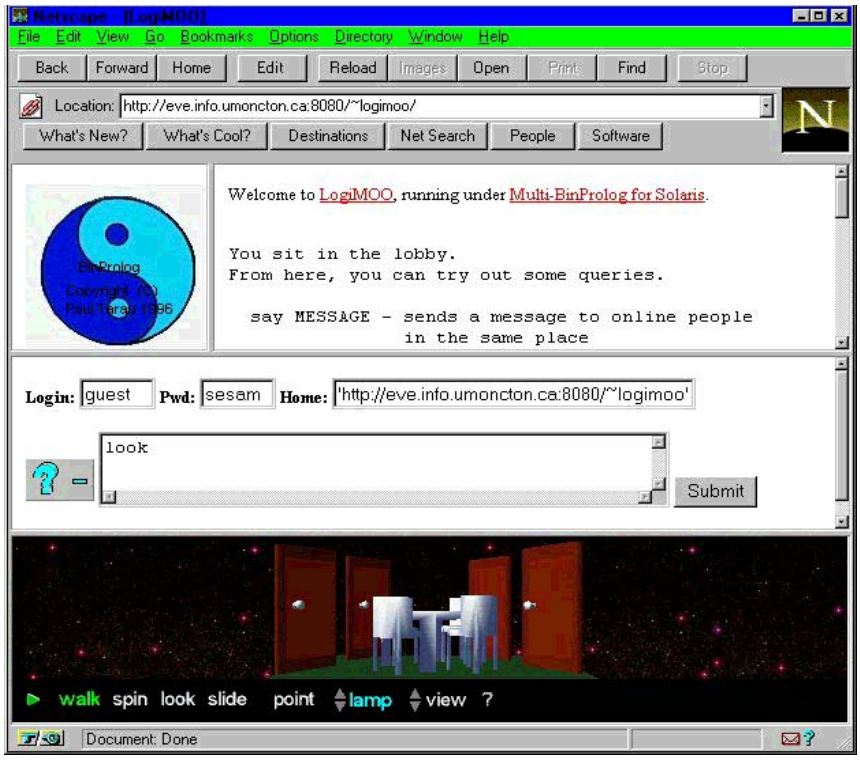

Figure 2: LogiMOO as a frame-based Netscape application

Clearly, achieving exclusively in Prolog or any other existing LP or CLP the equivalent of what we have developed in about 1-man/month total programming time would require a significant effort. We are more and more convinced that embedding logic programming tools in a multi-paradigm environment can compensate for their lack of advanced visual and INTERNET programming abilities and, ultimately, make them competitive for commercial development despite their small 'market share'.

As an embedded application LogiMOO acts as a broker between various multi-paradigm/multi-media Netscape components. It therefore keeps (a minimum amount of) state and user information. Its full Prolog command language gives arbitrary extensibility through objects and agents. Although file transfers and various protocols are implementable with the underlying Multi-BinProlog system, we have chosen to represent non-symbolic objects as hyper-links towards their owner's WWW home. Our design philosophy was to duplicate as little existing components as possible while achieving as much functionality as possible. At some point, we expect that LogiMOO will grow by itself through user extensions, much more than our own development effort, as a truly open virtual world, together with its present and future VRML and Java-centric cousins.

5 Related work

We have found out, after finishing the design and part of the implementation of LogiMOO that a pure representative of a programming paradigm (prototype-based OOP in this case) like SELF [US87] combined with a MOO-like visual interface [Smi95] has a lot in common with our objectives and motivations.

We do not know about other systems using logic programming for virtual world simulation, although a large number of sophisticated Web-based applications and tools are on the way to be implemented in LP/CLP languages, for instance [CH96a, LD96, CH96b, BBST96, SMS96, LDL96]. The closest application with a clear virtual world flavor is the Ubique Doors (tm) server [Sha94] which shows (Flat Concurrent) Prolog lists in log files although we do not know exactly how closely it is based on LP technology. This server, combined with the Sesame (tm) client emulates co-presence and cooperative work at virtual places implemented on top of existing Web pages and ftp directories. On the other hand applications of MOO technology usually combined with VRML navigation are spreading quite fast. Among them, some of the most impressive are:

- Intel's Moondo, http://www.intel.com/iaweb/moondo,
- Sony's Cyber Passage, http://vs.sony.co.jp/VS-E/vstop.html,
- Black Sun's CyberGate, http://www2.blacksun.com/beta/c-gate ,
- Netscape's CoolTalk+Live3D, http://home.netscape.com ,

- Worlds Inc.'s WorldChat, http://www.worlds.net ,
- SenseMedia's The Sprawl, http://sensemedia.net/sprawl .

Moreover, other declarative languages are starting to be used for WWW applications. The Carnegie Mellon FoxNET project [HL94] implements a full featured Web server. Microsoft's Active VRML proposal promises a declarative (purely functional) description of 3D movement and behavior.

6 Future work

The happy marriage between the key characteristics of a MOO and the tools offered by a Prolog implementation reinforced with simple but smoothly integrated coordination primitives of Linda (Multi-BinProlog) shows that logic programming is competitive for the next generation of software systems which tend to converge towards the same apex: programming as virtual world simulation.

Although not yet present in any implementations we know of, we believe that prototype-based OO in LP is more likely to take over than currently attempted class-based OO proposals. The main reason is that sharing vs. copying can be very cleanly described in terms of quantifiers and basic state and scope representation constructs like linear and intuitionistic implications. Our object system, inspired by MOO functionality and avatar programming has a simple (and largely language independent) Datalog-based model. Work on an object calculus formalizing its properties and an efficient distributed transitive closure maintenance algorithm upon updates is in progress. This is expected to be subject of extensive future work on top of LogiMOO.

Optimization of the CGI interface, as well as full integration with HTML and VRML are also high priority future developments at the implementation level.

The major interactive feature LogiMOO currently lacks is a concept of active Web page with realtime notification. Ubique's Sesame/Doors system solves this with a dedicated parallel server and a customized Mosaic browser. This solution is not practical for us as we want to have an application which runs under the user's own Netscape (or compatible) browser and therefore benefiting from third party VRML plug-ins. We are working on a client side Java/JavaScript solution to this problem as we want to avoid costly 'server–push' or 'client–pull' approaches.

The development a security scheme is also needed for realistic uses of LogiMoo. An important aspect is related to resource control. Although Multi-BinProlog has built-in protection against data area overflows, it has no timeout based computational ressource control. A timeout mechanism, similar to the one in SICStus Prolog is planned for the next release. On the other hand, using craft/1, for example, a given user can put an arbitrary number of objects on

a server. We plan to limit this either through the amount of data that can flow through a virtual blackboard or by an object quota mechanism as in other MOOs. Some of these problems are naturally solved with the "web of MOOs" scenario, where each user monitors resource uses on his own computer at OS level, while allowing others "read" access, as usual on the WWW, to objects in his `public_html` directory.

7 Conclusion

We have shown that Prolog with appropriate coordination language extensions is a practical tool for virtual world simulation. A synergy between MOOs, Linda-style coordination and Prolog's powerful associative search and dynamic object creation facilities could be expected. Multi-BinProlog's threads and virtual blackboards make this synergy possible. A logic programming approach to MOO programming has the advantage of having all the right tools within a unified environment. Embedding in Netscape ensures implicit platform independence of our server and seamless cooperation with present and future third party Netscape tools.

Acknowledgement

Paul Tarau thanks for support from NSERC (grant OGP0107411), from the FESR of the Université de Moncton. Koen De Bosschere is senior research assistant with the Belgian National Fund for Scientific Research. Special thanks go to Daniel Perron for long discussions helping to come out with the initial idea of LogiMOO. Mathieu Brideau, Hoang-Anh Nguyen, Stephen Rochefort were among the first virtual inhabitants of LogiMOO and have helped with various aspects of its design and its implementation. We also thank the referees for their valuable suggestions and comments.

References

[BBST96] Ph. Bonnet, L. Bressnan S., Leth, and B. Thomsen. Towards ECLIPSE Agents on the Internet. In Tarau et al. [TDDBH96]. http://clement.info.umoncton.ca/ lpnet.

[CH96a] D. Cabeza and M. Hermenegildo. html.pl: A HTML Package for (C)LP systems. Technical report, 1996. Available from http://www.clip.dia.fi.upm.es.

[CH96b] D. Cabeza and M. Hermenegildo. The Pillow/CIAO Library for Internet/WWW Programming using Computational Logic Systems.

In Tarau et al. [TDDBH96]. http://clement.info.umoncton.ca/ lpnet.

[DBJ93] Koen De Bosschere and J.-M. Jacquet. Multi-Prolog: Definition, Operational Semantics and Implementation. In *Proceedings of the ICLP'93 conference*, Budapest, Hungary, 1993.

[DBPT96] Koen De Bosschere, Daniel Perron, and Paul Tarau. LogiMOO: Prolog Technology for Virtual Worlds. In *Proceedings of PAP'96*, pages 51–64, London, April 1996. ISBN 0 9525554 1 7.

[DBT93a] K. De Bosschere and P. Tarau. Blackboard Communication in Logic Programming. In *Proceedings of the PARCO'93 Conference*, pages 257–264, Grenoble, France, September 1993.

[DBT93b] K. De Bosschere and P. Tarau. Some Issues in the Implementation of a Unix-based Blackboard. In K. De Bosschere, J.M. Jacquet, and P. Tarau, editors, *Proceedings of the ICLP'93 Post-Conference Workshop on Blackboard-Based Logic Programming*, pages 91–104, Budapest, Hungary, June 1993.

[DBT96] K. De Bosschere and P. Tarau. Blackboard-based Extensions in Prolog. *Software — Practice and Experience*, 26(1):49–69, January 1996.

[HL94] Robert Harper and Peter Lee. Advanced Languages for Systems Software. Technical report, 1994. CMU-CS-FOX-94-01.

[LD96] S. W. Loke and A. Davison. Logic programming with the world-wide web. In *Proceedings of the 7th ACM Conference on Hypertext*, pages 235–245. ACM Press, 1996.

[LDL96] S. W. Locke, A. Davison, and Sterling L. Lightweight Deductive Databases for the World-Wide Web. In Tarau et al. [TDDBH96]. http://clement.info.umoncton.ca/ lpnet.

[Sha94] Ehud Shapiro. Enhancing the WWW with Co-Presence. In *Proceedings of the 2nd International Conference on the WWW*, 1994.

[Smi95] D. Smith, R.B. Maloney J. Ungar. The self-4.0 user interface: Manifesting a system-wide vision of concreteness, uniformity, and flexibility. In *Proceedings OOPSLA '95, ACM SIGPLAN Notices*, 1995.

[SMS96] Peter Szeredi, Katalin Molnár, and Rob Scott. Serving Multiple HTML Clients from a Prolog Application. In Tarau et al. [TDDBH96]. http://clement.info.umoncton.ca/ lpnet.

[TDDBH96] Paul Tarau, Andrew Davison, Koen De Bosschere, and Manuel Hermenegildo, editors. *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP'96, Bonn, September 1996. http://clement.info.umoncton.ca/ lpnet.

[US87] D Ungar and R.B. Smith. Self: The Power of Simplicity. In *Proceedings OOPSLA '87, ACM SIGPLAN Notices*, pages 227–242, 1987.