Controlling Virtual Worlds Through Extensible Natural Language

Stephen Rochefort and **Veronica Dahl**
Logic and Functional Programming Group
School of Computing Science
Simon Fraser University
{srochefo,veronica}@cs.sfu.ca

Paul Tarau
Département d'Informatique
Université de Moncton
tarau@info.umoncton.ca

Abstract

LogiMOO is a BinProlog based Virtual World that provides methods for Internet users to create virtual places, virtual objects, and intelligent agents. The original design of the LogiMOO kernel provided an interactive Prolog syntax interface. In this paper, we present a LogiMOO Natural Language interface based upon Assumption Grammar and Hidden Accumulator theory. This paper is electronically available at http://www.cs.sfu.ca/cs/people/GradStudents/srochefo/personal/papers/publications.html.

Introduction

Virtual Worlds are a strong unifying metaphor for various forms of net-walk, net-chat and INTERNET-based virtual presence in general. Some fairly large-scale projects (Intel's Moondo(Intel 1997), Sony's Cyber Passage (Sony 1997), Black Sun's CyberGate (BlackSun 1997), Netscape's CoolTalk+Live3D, Worlds Inc.'s WorldChat) and Multi-User Dungeons (MUDs) and Muds Object-Oriented (MOOs), such as Avalon MUD (Avalon 1997), converge 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.

Presently, despite their graphical sophistication, virtual worlds do not allow *controlling* behaviour and object creation (i.e. "programming with words"). Yet their characteristics favour the use of natural language: each virtual world represents a particular domain of interest, so that its associated relevant subset of language is naturally restricted; and the command language into which natural language sentences would have to be parsed is formal and straightforward enough while being already relatively close to natural language.

In this article we incorporate natural language consultation capabilities into a virtual world system, LogiMOO, that was introduced in (De Bosschere, Perron, & Tarau 1996). Our natural language front end is extensible in the sense that it is easy for the user to tailor it to a particular virtual world through defining the nouns, verbs and adjectives specific to that world. These definitions are done in user-friendly terms, with the system being responsible for their correct and efficient integration into the rest of the grammar, and for their correct and efficient correspondence with LogiMOO objects and actions.

Our natural language component illustrates as well an interesting use of Assumption Grammars (Tarau, Dahl, & Fall 1995; Dahl *et al.* 1996), which are basically like Definite Clause Grammars except that they can handle multiple accumulators invisibly, and that they possess linear and intuitionistic implications scoped over the current continuation. In this paper we also show how Assumption Grammars can simplify the treatment of some crucial computational linguistics problems (e.g. long distance dependencies), while simultaneously facilitating more readable grammars.

The LogiMOO Kernel

Objects in LogiMOO[1] are represented as hyper-links (URLs) towards their owners' home pages where their "native" representation actually resides in various formats (HTML, VRML, GIF, JPEG etc.).

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.

LogiMOO's primitive operations are implemented on top of Multi-BinProlog's Linda-style operations (Tarau 1996). We refer to (De Bosschere, Perron, & Tarau 1996; Tarau & De Bosschere 1996) for a full description of these operations. Here are the basic primitives:

out(X)	Puts X on the server.
in(X)	Takes an object matching X from

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

	the server.
all(X,Xs)	Reads the list Xs matching X currently on the server.
run(Goal)	Starts a background process executing Goal.
local_out(X)	Puts private information X on the local default blackboard.
local_rd(X)	Checks whether an object matching X is on the local blackboard.

A number of derived operations are built on top of the primitive LogiMOO operations.

rd(X)	Deterministic read.
cout(X)	Conditional out: Puts X on the server unless an object matching X is found on the server.
cin(X)	Conditional in: Takes an object matching X from the server and fails if no such object is found.
forall(X,G)	Executes goal G for all objects on the server matching X.

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 a 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)).
```

Communication in LogiMOO

The notifier is one of the simplest possible *agents*. It is automatically started from the login predicate as a background thread with `run(notifier(Name))`. The notifier's thread blocks until a message to Name arrives, at which the notifier simply outputs the message.

`where(Place)` unifies `Place` with the location of the user's avatar. To conditionally create a place (unless it exists) we use: `place(Place)`. To create a port (only when the links are already existing places), we use: `port(P1,Dir,P2)`. To "Teleport" O from P1 to P2, we use: `move(O,P1,P2)`.

On top of teleporting, we implement `go(Dir)`, `craft(O)` (which creates virtual objects with ownership), and `give(Who, What)`.

`Look/O` 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 created by a given user are shown as links relative to the user's home page.

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.

Assumption Grammars

Basically, we can consider Assumption Grammars to be a syntactic variant of Definite Clause Grammars augmented with hypothesizing capabilities. Their main features are:

- terminals (words) are noted `#Word` rather than `[Word]`;
- rewriting is denoted `:-` rather than `-->` .
- linear assumptions (noted `+Assumption`) over the current continuation (i.e. the remaining AND branch of the resolution) temporarily add a clause usable *at most once* in later proofs. They are consumed by their counterpart call, -Assumption.
- intuitionistic assumptions are similar but the clauses are reusable *an indefinite number of times*

In rigor, the syntactic modifications involve more than just syntactic sugar. In particular, we have invisible multiple accumulators which are accessible through built-ins such as:

```
dcg_def/1   % sets the first invisible DCG
            % argument.
dcg_val/1   % retrieves the current state of the
            % DCG stream
```

For the purposes of this paper we shall use only one invisible accumulator to automate the handling of the input and output string. The values of these strings will be then defined and consulted, respectively, through dcg_def/1 and dcg_val/1. A more complete description of Assumption Grammars can be found in (Tarau, Dahl, & Fall 1995).

The Natural Language Interface to LogiMOO

Since LogiMOO handles mostly commands, imperative sentences will be the most common ones. In these, the subject (the avatar that the user is controlling) is usually left implicit. The user first enters a set of imperative subjectless sentences into the Netscape input field, and this input is sent through the parser to be converted into LogiMOO kernel predicates which are then executed to complete the actions. The complete parser, an overview of which we next present, is available from the authors upon request.

Netscape Interface

All input received, which comes in the form of a list of ASCII codes, is taken from the user through a Netscape interface. This input includes the user's login name and password to the virtual world, a home page uniform resource locator (URL) and the NL input. From all this, the input handler extracts the NL input as a separate item, which is then sent to the parser after its conversion into a list of words.

The list of words obtained is then translated by the parser into a list of predicates, as we shall describe next.

The Lexicon

Avatars, Objects and Places - Noun Definitions Information about the avatars controlled by the users and the virtual objects and virtual places that have been created can be provided in noun definitions.

For instance, avatars are introduced by proper names. The rule:

(3) proper_name(john-[masc,sing]) :- #john.

defines the word john as a masculine and singular proper name represented by the constant john.

Objects are introduced by noun definitions, e.g.

(4) noun(car-[neut,sing]) :- #car.

Virtual places are also introduced by nouns, and are always set to a neutral gender and singular form, e.g.

(5) noun(south-[neut,sing]) :- #south.

Actions - Verb Definitions Verbs in the LogiMOO environment represent actions that can take place in the virtual world.

Intransitive verbs correspond to actions performed by an avatar her/himself and involve no other specific avatar, place, or object, e.g.

(6) intrans_vb(smile) :- #smile.

Transitive verbs involve one other avatar, object or place in the the virtual world. For instance, the rule:

(7) trans_vb(Y,go(Y)) :- #go.

corresponds to a user actioning her/his avatar to go someplace in the virtual world. The two arguments identify the place, *Y*, and the translation to the unary predicate *go*.

Bitransitive verbs specify an action by an avatar that involves any two objects, avatars, or places. As an example,

(8) bitrans_vb(Y,Z,give(Y,Z)) :- #give.

specifies the action of the avatar giving some other avatar, *Y*, some object, *Z*. The third argument is the predicate, *give(Y,Z)*, to be executed by the LogiMOO kernel.

Notice that in all verbs, the subject is left implicit, given that the LogiMOO kernel recognizes it as the avatar who logged in.

Pronouns, Determiners, and Prepositions Pronouns are specified in a similar way as nouns:

(9) pronoun(_X-[fem,sing]) :- #she.

This specification identifies a pronoun, *she*, with a feminine gender and singular form. Agreement information is used to resolve pronoun references into the correct avatar or virtual object or place.

Determiners and prepositions are specified as

(10) det :- #the.
(11) preposition :- #to.

Syntactic Rules

As we have seen, all sentences are in imperative form, with their subject left implicit. Thus they reduce to verb phrases, which can be of the following forms:

(VP1) An intransitive verb.
(VP2) A transitive verb followed by a noun.
(VP3) A transitive verb followed by a noun phrase.
(VP4) A transitive verb followed by a prepositional phrase.
(VP5) A bitransitive verb followed by two noun phrases.
(VP6) A bitransitive verb followed by a noun phrase and a prepositional phrase.

A prepositional phrase is defined as

(PP1) A preposition followed by a noun phrase.

The noun phrase forms allowed are

(NP1) A proper name.
(NP2) A pronoun (anaphora).
(NP3) A determiner followed by a noun.

(more complex noun phrases will be explained in the next section)

In addition, we identify communication inputs which occur when a user wants their avatar to say, whisper or yell some message, e.g.

say hi how are you.

This form of input is introduced by either:

(F1) The word "whisper" followed by a prepositional phrase followed by a message.
(F2) The word "say" followed by a message.
(F3) The word "yell" followed by a message.

Table 1 shows some sample parses.

Treating Dependencies Through Assumption Grammars

Single Sentence Dependencies-Relativization

Consider the noun phrase "the review that John saw". The relative clause in it, "that John saw", can be viewed as a sentence in which the object noun phrase required for "saw" is missing (to be equated with the antecedent "the review"). The dependency between the antecedent and the missing noun phrase is called "long distant" because any number of words can intervene between them (cf. "the review that the new faculty member saw", and so on).

Following (Hodas 1992; Hodas & Miller 1994) we can use linear logic to temporarily augment a grammar with a missing noun phrase rule, only during the parse of a relative clause:

NL Input	Translation	LogiMOO Action
look.	look.	Provides the user with a description of the room that their avatar currently occupies.
craft a car.	craft(car).	Creates a virtual object, car, owned by the avatar.
craft a car. give it to john.	craft(car), give(john, car).	Creates a virtual object, car, and gives it to john.
take the car that john crafted.	and(take(X), crafted(john, X), is_a(X, car))	Puts a car object crafted by john into the avatar's possession.
whisper to john	whisper(john, 'How are you').	Sends the message 'How are you' to john.

Table 1: Sample Parses

```
rel(X,Pred,and(Pred,R)):- #that, +missing_np(X)-:
                          sent(R).

np(X):- -missing_np(X).
```

The first of these rules recognizes a relative clause as a sentence introduced by the word "that", under the linear assumption that there is an (implicit) noun phrase with meaning representation X in the scope of "sent".

The second rule accepts an assumed missing noun phrase, together with its meaning X, as one of the possible manifestations of a noun phrase.

Figure 1 exemplifies through the derivation of a relative clause. The assumption is noted inside a box, and it's consumption is noted by the arrow relating it to the assumed predicate which contains it as an argument.

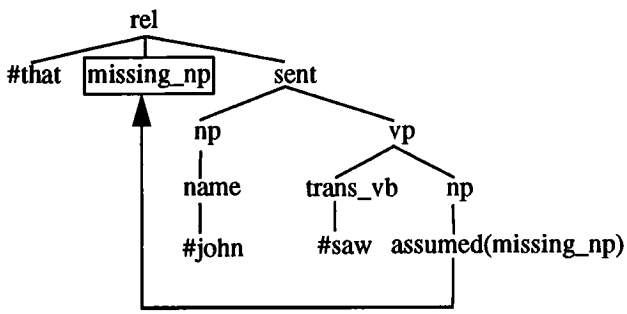

Figure 1: Relativization with assumptions

Discourse Dependencies - Anaphora

An interesting example of the use of linear implication is the way it can deal with inter-sentential dependencies such as anaphora, in which a given noun phrase in a discourse is referred to in another sentence (e.g. through a pronoun). We refer to the noun phrase and the pronoun in question as entities which co-specify, since they both refer to the same individual of the universe.

As a discourse is processed, the information gleaned from the grammar and the noun phrases as they appear can be temporarily added as hypotheses ranging over the current continuation. Consulting it then reduces to calling the predicate in which this information is stored.

We exemplify the hypothesizing part through the following noun phrase rules:

```
np(X,VP,VP):- proper_name(X),
   +(specifier(X)).
np(X,VP,R):- det(X,NP,VP,R),
   noun(X-F,NP), +(specifier(X-F)).

pronoun(X-[masc,sing]):- #he.
pronoun(X-[fem,sing]):- #her.

anaphora(X):- pronoun(X).

noun(X-[fem,sing],woman(X)):- #woman.
```

The linear assumption, +(specifier(X)), keeps in X the noun phrase's relevant information. In the case of a proper name, this is simply the constant representing it plus the agreement features gender and number; in the case of a quantified noun phrase, this is the variable introduced by the quantification, also accompanied by these agreement features. The pronoun rules keep track of gender and number features and the noun rules do the same plus they construct, from a variable X and the noun in question, a predication standing for its meaning.

Potential co-specifiers of an anaphora can then consume the most likely co-specifiers hypothesized (i.e., those agreeing in gender and number), through a third rule for noun phrase which uses a hypothesized specifier upon encountering an anaphora:

```
np(X,VP,VP):- anaphora(X), -(specifier(X)).
```

Semantic agreement can be similarly enforced through the well-known technique of matching syntactic representations of semantic types.

This methodology can of course be extended in order to incorporate subtler criteria. For instance, we can make each pronoun carry, at the end of the analysis, the whole list of its potential referents as a feature. User-defined criteria can then further refine the list of candidate co-specifiers, as in (Fall, Dahl, & Tarau 1995).

It is interesting to point out that in order to handle abstract co-specifiers (Asher 1993) such as events or propositions, all we have to do is extend the definition so that other parts of a sentence can be identified as possible specifiers as well. For instance, for recognizing "John kicked Sam on Monday" as the co-specifier of "it" in the discourse: "John kicked Sam on Monday. It hurt.", we can simply make the linear assumption that sentences are potential co-specifiers for pronouns of neutral gender.

A Methodology for Extension

As mentioned in the introduction, we can tailor our language interface to a particular virtual world by simply adding lexical definitions of the most relevant words corresponding to this world's domain. These are typically: nouns, verbs and adjectives.

In order for this addition to be performed in a user-friendly way, we are currently working on an application that provides the user with a sample word from each type of an extensible syntactic category (e.g. "smile" as a sample intransitive verb, "look" as a sample transitive verb, "give" as a sample bitransitive verb, etc.). The user then only needs to a) enter the new word, b) click on one of the sample words displayed to indicate that the new word is "like the clicked one" (i.e., of same syntactic category), and c) provide a corresponding predicate name, as well as the predicate's definition which will be added in the LogiMOO component of our system. Thus the user need not even be aware of the names of syntactic categories, but the analogy with existing sample words allows our interface to invisibly replicate a similar lexical definition. For instance, if the user enters "take" as a new word and by clicking on a sample similar word identifies it as a transitive verb, and gives us a corresponding predicate name, the following lexical definition will be automatically created:

```
trans_vb(Y, take(Y)) :- #take.
```

A definition for the corresponding predicate is:

(S4) take(O) :-
whoami(I),
where(Place),
cin(contains(Place,O)),
out(has(I,O)).

Of course, this technique can equally serve to extend a grammar with other types of words than nouns, verbs and adjectives. However, we have chosen to focus on just these categories because they are the most likely to be application-dependent, and because they are the ones that will induce corresponding new commands in the LogiMOO component of our system.

Related Work

We have not found so far any MUD/MOO environments that handle NL processing. Other MUD/MOO environments fall into two general categories. Environments such as "Moondo" by Intel (Intel 1997), "CyberGate" by BlackSun (BlackSun 1997), and "Cyber Passage" by Sony (Sony 1997), fall into the category of point and click graphical environments. These completely avoid the need for NL processing as the only text involved seems to be that for chatting with other avatars. All movement and actions are completed with mouse point and click actions. The second category, in which environments such as MediaMOO (MediaMOO 1997) and the Avalon MUD (Avalon 1997) fall into, are textual based systems. These systems lack NL processing and focus on the use of pattern matching techniques to gather information.

Further, there is little work being done that is specific to the advantages gained by connecting MUDs/MOOs to the World Wide Web using logic programming such as the use of Prolog. As such, objects in LogiMOO are represented as hyper-links (URLs) towards their owners' home pages where their 'native' representation actually resides in various formats (HTML, VRML, GIF, JPEG, etc.). At the same time, logic programming adds deductive database facilities in a uniform framework, hypothetical reasoning tools (through Assumption Grammars), and logic programming data and code use the same representation which makes meta-programming easy.

Conclusion

We have presented a NL interface to LogiMOO which takes a controlled form of English and translates it into LogiMOO kernel predicates which are executed as actions in the virtual environment. Pronominal references in multisentential input are allowed.

Basing the interface implementation on Assumption Grammar and Hidden Accumulator theory, we reduce the overhead associated in other DCGs with list maintaining and reference resolution. This leads to a very simple method of building a larger translation base.

Compared to other currently known MUD/MOO environments, this interface bridges the gap between those that are graphical based and those that are pattern matching based. By filling the gap, we are able to provide the users with a natural form of textual interaction on which graphical environments can still be built.

Although the current interaction is controlled completely through the natural language interface, this does restrict efficiency of maneuvering an avatar through the virtual worlds. Future work will move to-

wards including some basic visual navigation to complement the natural language interface.

Finally, the LogiMOO system is currently being used in some undergraduate classes as an enhancement to the class and as a testbed for LogiMOO. Future directions are to include LogiMOO as a tool for virtual tele-education where distance education students and instructors may use LogiMOO as a learning/teaching environment.

Acknowledgements

We are thankful for the support from NSERC (Paul Tarau's grant OGP0107411 and Veronica Dahl's grant 31-611024) and from the FESR of the Université de Moncton.

References

Asher, N. 1993. *Reference to Abstract Objects in Discourse*, volume 50 of *Studies in Linguistics and Philosophy*. Kluwer.

Avalon. 1997. The Avalon MUD. http://www.avalon-rpg.com/.

BlackSun. 1997. CyberGate. http://www.blacksun.-com/.

Dahl, V.; Fall, A.; Rochefort, S.; and Tarau, P. 1996. A Hypothetical Reasoning Framework for NL Processing. In *Proc. 8th IEEE International Conference on Tools with Artificial Intelligence.*

De Bosschere, K.; Perron, D.; and Tarau, P. 1996. LogiMOO: Prolog Technology for Virtual Worlds. In *Proceedings of PAP'96*, 51–64. ISBN 0 9525554 1 7.

Fall, A.; Dahl, V.; and Tarau, P. 1995. Resolving co-specification in contexts. In *Proc. of IJCAI Workshop on Context in Natural Language Processing.*

Hodas, J. S., and Miller, D. 1994. Logic Programming in a Fragment of Intuitionistic Linear logic. *Journal of Information and Computation* 110(2):327–365.

Hodas, J. 1992. Specifying Filler-Gap Dependency Parsers in a Linear-Logic Programming Language. In Apt, K., ed., *Logic Programming Proceedings of the Joint International Conference and Symposium on Logic programming*, 622–636. Cambridge, Massachusetts London, England: MIT Press.

Intel. 1997. Moondo. http://www.intel.com/iaweb/-moondo/index.htm.

MediaMOO. 1997. MediaMOO. telnet://guest@purple-crayon.media.mit.edu:8888/.

Sony. 1997. Cyber Passage. http://vs.sony.co.jp/VS-E/vstop.html.

Tarau, P., and De Bosschere, K. 1996. Virtual World Brokerage with BinProlog and Netscape. In Tarau, P.; Davison, A.; De Bosschere, K.; and Hermenegildo, M., eds., *JICSLP'96.*

Tarau, P.; Dahl, V.; and Fall, A. 1995. Backtrackable State with Linear Assumptions, Continuations and Hidden Accumulator Grammars. In *ILPS'95 Workshop on Visions for the Future of Logic Programming.*

Tarau, P. 1996. BinProlog 5.25 User Guide. Technical Report 96-1, Département d'Informatique, Université de Moncton. Available from *http://clement.info.umoncton.ca/BinProlog.*