Ossa - A Conceptual Modelling System for Virtual Realities

Finnegan Southey and James G. Linders

[1] Dept. of Computer Science, University of Waterloo Waterloo, Ontario, Canada, N2L 3G1 fdjsouthey@uwaterloo.ca

[2] Dept. of Computing and Information Science University of Guelph, Guelph, Ontario, Canada, N1G 2W1 jgl@snowhite.cis.uoguelph.ca

Abstract. As virtual reality systems achieve new heights of visual and auditory realism, the need for improving the underlying conceptual modelling facilities becomes increasingly apparent. The Ossa system provides a media-independent modelling environment based on a production system model that uses conceptual graphs to represent both the facts and the rules. Using conceptual graphs allows for interaction with the virtual world using multiple modalities (e.g. graphics and natural language). Conceptual graphs also allow for highly expressive facts and rules, and a diagrammatic programming technique. The motivation, design, and implementation of the Ossa system are discussed.

1 Introduction

In recent years virtual reality (VR) research has produced incredible sensory realism for large and complex worlds. In scientific visualization, engineering, education, and entertainment it is making rapid advances. The construction of virtual worlds may be divided into two broad areas: presentation and modelling. The presentation domain is primarily concerned with the rendering and interface presented to the user. The modelling domain is concerned with the simulation of behaviours and relationships amongst objects. It focusses on representing and manipulating knowledge about virtual worlds, often involving a variety of broad domains.

On the presentation side, significant progress has been made in producing realistic graphical and auditory environments. User interfaces have also improved, although not to the same degree. However, on the modelling side there has been less progress, except within specific sub-areas. For example, numerical modelling of physical laws has seen substantial advances, but the conceptual modelling of arbitrary relationships between objects has been neglected. This leads to virtual worlds with sensory verisimilitude and convincing physical behaviour but with "shallow" causal relationships and limited, often "scripted", interaction between virtual agents and objects.

To address these shortcomings, we present the design and implementation of Ossa [1], a conceptual modelling system for the development of virtual worlds which employs conceptual graphs (CG's) as its key method for representing knowledge and employs a production system to handle the world dynamics.

2 Background

2.1 Conceptual Modelling in Contemporary VR

Among the earliest examples of conceptual modelling that may be related to modern virtual reality are those found in traditional planning and robotics research. These problem domains involve reasoning about a model of the real world (e.g. STRIPS [2]). In such models, an agent interacts with the world producing a sequence of changes in that world, often with some sort of feedback to the agent. In selecting this comparison, we seek to introduce a broader conception of the term "virtual reality" than is popularly accepted. We will here state a simple definition to clarify our position:

> *virtual reality*: a computer-based model of some aspects of reality including the virtual presence of an agent that can perceive and interact with the model

We do not assert that this definition is more or less appropriate than any other definition. It is only provided to delimit the scope of our discussion. Note that we place no restrictions on how the virtual world may be presented or manipulated. We also state only that "some agent" interacts with the world. While the agent may often be human, it could be non-human (e.g. an animal employed in a pyschological study or a robot being trained or tested). We specify an interactive world to distinguish virtual realities from the broader class of "simulations".

With this definition in hand, we can again consider the kind of virtual worlds that bear a resemblance to early planning research. While not directly inspired by that research, a significant body of text-based virtual realities exist that offer similar conceptual models of reality, focussing on relationships between objects rather than on numerical simulation.

Originating in simple, single-user games and amusements, such systems underwent a rapid advancement in the early 90's forming a set of text-based, multi-user, internet-accessible virtual worlds commonly known as MUD's (from the name "multi-user dungeon") [3]. These systems originally described the world using only text and accepted interactions in the form of highly simplified natural language commands (e.g. "get the ball", "look", "give the apple to the moose"). More recently, people have experimented with adding two- and three-dimensional graphical interfaces [4][5], and are rapidly advancing towards the more popular conception of VR.

The conceptual modelling facilities offered in modern MUD's are, in most aspects, considerably more advanced than those offered by multimedia systems. With few exceptions (the most notable of which is LogiMOO [6]), the most powerful MUD's use an object-oriented (OO) language to build their conceptual models. The languages offered in LambdaMOO [7] and its like are very full-featured, offering a variety of features such as multiple-inheritance, polymorphism, garbage-collection, dynamic class-loading, and more. They have allowed the creation of highly sophisticated virtual worlds that deal in both concrete (e.g. walls, doors) and abstract (e.g. hostility, emotional state, desire) concepts.

2.2 Limitations of the Object-Oriented Approach

When exploring the design and construction of existing environments, we have found that the object-oriented model can serve as a hindrance to the ongoing expansion and refinement of a virtual world. Rome was not built in a day, and neither are complex virtual worlds. Their design is frequently revised over time to include new domains and new levels of detail. We found object-oriented models difficult to maintain in these circumstances, owing chiefly to the necessity of rewriting class interfaces or rearranging the inheritance hierarchy. The problems with object-oriented design in conceptual modelling and software design have been noted elsewhere [8][9], but we will briefly describe the chief problems for VR design here.

In many cases, relationships amongst entities are more plentiful and more complex than the entities themselves in a virtual world. The object-oriented model can complicate the implementation of relationships amongst entities, especially relationships involving many objects. Encapsulation can introduce unnatural and conceptually unnecessary boundaries into the description of a relationship. Such boundaries arise when the designer must decide how to divide responsibility for the relationship between classes (e.g. which classes should store the different pieces of information about the relationship). Such decisions can be complex and time-consuming, and subsequent revisions even more so.

Finally, object-oriented systems require an event model to detect changes in the world and respond to them. Events are typically modelled as messages originating from the object(s) involved in the event. It is difficult to foresee all the events that may be of interest in the future and the facilities for generating a given event must be explicitly provided. Adding support for events that may never be needed is wasteful but adding them at a later date may require extensive redesign.

In dealing with these issues we concluded that the best idea was to "promote" the role of relationships within the system to have a status equal to that of other entities and, at the same time, shed the encapsulation of the object-oriented approach while preserving inheritance capabilities. Additionally, we decided to replace the message-based event model with an execution model that can readily and flexibly detect and respond to any changing patterns (events) within the world's knowledge base. These two goals led us to the choose a production system for the execution model and conceptual graphs to fill the role of the basic unit of knowledge representation.

3 Alternate Approaches

3.1 Conceptual Graphs

Conceptual graphs offer an attractive set of features for knowledge representation. First and foremost, relations and concepts have an essentially equal footing in CG's. Relations have no constraints on their valence other than those imposed

by the designer, and have a type hierarchy of their own allowing for specialization/generalization, a key feature offered by the object-oriented approach. Furthermore, the use of nested contexts allows for a great deal of expressive power while retaining some of the benefits found in an OO language's encapsulation.

In terms of interface modalities, CG's are attractive because of their close relationship with natural language processing. This means that natural language interfaces for designing or interacting with a virtual world are straight-forward compared to the object-oriented approach, and by separating the semantic information in the CG's from the actual words used, descriptions of the world may be rendered into several human languages. CG's are also mathematically formalized, which allows their expression using formal languages and mathematical formulae.

There is a diagrammatic representation of CG's which has reasonably wide acceptance. This allows for diagrammatic programming of the virtual world. While diagrammatic programming can easily become cumbersome in large systems, it may provide a useful way to browse the VR system. Diagrams may also offer a gentler learning curve to new programmers of the system (in the authors' experience, it is not uncommon for members of a VR user community to participate in development efforts and occasionally progress from complete novices to competent programmers).

Finally, work has been done on constructing three-dimensional graphical models[10] from CG's. While still in its infancy, this work is very important for using the world model in multimedia environments.

We envision that a CG-based virtual world would offer multi-modal interaction and rendering. An architectural project could begin with two-dimensional schematics of a building constrained by physical laws represented through mathematical formulae. The architect could then navigate through a three-dimensional rendering of the building and use simple verbal and gestural instructions to make minor alterations. Even details such as decoration and furniture layout could be included in the virtual specification. Naturally, this is far beyond our current reach, but we regard this as the goal towards which we should strive and which is best served by using an interface-independent knowledge representation like conceptual graphs.

3.2 Production Systems

In object-oriented VR work, there is a great tendency to "script" sequences of actions in a deterministic and fairly rigid manner. This is a natural side-effect of using the procedural programming approach. The result is a world in which only prescribed chains of events occur. This situation can be improved but it requires the creation of a sophisticated and time-consuming messaging event model that offers sufficient detail in its events to allow for a wide variety of "cause-and-effect" relationships.

Another popular means for describing dynamics is a *Petri net* [11]. While these provide a clean representation for a fully described set of processes, and moreover have a mapping to CG's, it is not immediately obvious how they can

be applied without giving rise to the same "scripting" effect described above. Directly mapping an entire world state to a subsequent state is precisely what we strive to avoid. Rather, we wish to allow several distinct aspects of the world state give rise to several distinct consequences in an approximately parallel fashion, as occurs in the real world. Only a small portion of the world's state would provide the basis for the kind of simple cause-and-effect events we are attempting to capture, and so the Petri net's enumeration of the relevant states would be complicated and contrary to our goals.

The choice of a production system to provide the execution model stems directly from this simple view of "cause-and-effect" dynamics. Some subset of the world's state constitutes a cause which automatically gives rise to a change in the world state (an effect). Rather than explicitly scripting chains of events, the dynamics may be factored into simple pre-/post-condition pairs.

We believe that the production system approach offers this complexity of cause-and-effect in a relatively straight-forward fashion. Effects and their causes are expressed using production system rules. Adding a new cause-and-effect relationship is simply a matter of adding a new rule. Since a production system typically has unlimited access to its knowledge base, these rules can be based upon any set of facts the programmer chooses. This allows for virtually unlimited development of the system, unfettered by the need to elaborate an event model. Also, since production rules are independent of the facts they deal with, changing behaviours requires only the modification of a small number of rules, unlike the object-oriented approach where methods are bound to data structures and interfaces in such a manner that changes are often difficult or costly to implement.

4 The Ossa System

4.1 The Architecture of Ossa

Having identified CG's as our knowledge representation and production systems as our execution model, it was decided to fuse the two by using CG's to represent the facts and rules of a production system. Similar work has been done in the past within the CG community [12][13][14]. In our case, we decided to maximize the expressive power of our facts and rules by allowing the use of compound CG's. This can improve efficiency in the system by hiding details within nested graphs and by making rules that deal with these more focussed contexts. The result is the Ossa system. The name is taken from the Latin word for "skeleton", representing our belief that VR systems should have a strong conceptual model at the core of their construction.

Ossa is built using the three-layer architecture shown in Figure 1. The horizontal layers represent the three software layers which comprise the system. The two columns on the right represent the conceptual division of each layer into the two components of a production system, the working memory and the knowledge base (or rule set). The downward arrow represents the downward dependence

of the layers, with each layer depending only on the layer below it and knowing nothing of the layer above it.

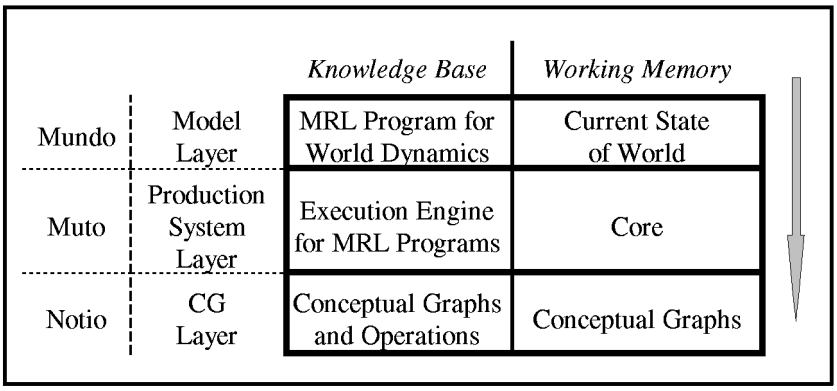

Fig. 1. Architecture of Ossa

The bottom layer, which is called the Notio layer, provides the raw CG representation and handling ability. It is the reference implementation of our Notio API [15]. It provides facilities for constructing, manipulating, and matching CG's. It also provides a Conceptual Graph Interchange Format (CGIF) parser and generator for input and output. It is a general-purpose API which is flexible enough to be used for Ossa. From the production system perspective, CG's in Ossa form the facts of the working memory and are used in the construction and evaluation of rules for the knowledge base.

The middle layer, the Muto layer, provides the basic production system capabilities of the Ossa system. It is responsible for managing the CG's which make up the working memory, and for parsing and evaluating the rules that make up the knowledge base. It uses the classes and methods of the Notio layer to accomplish these tasks. It is divided into two basic parts. The *core* is essentially the working memory. It stores the CG's that are the facts describing the current state of the virtual world. The other part is the execution engine which evaluates and executes the production rules. It is called the *production agent* and operates by parsing and executing a set of rules written in the Muto Rule Language (MRL) . The MRL is described in greater detail below. The overall architecture of the Muto layer is shown in Figure 2.

The topmost layer is the model layer, which we have called the Mundo layer. This is the layer which is used to describe the conceptual model of the virtual world. It is created by writing a set of MRL rules that describe the dynamics of the world. There is typically a "bootstrap" rule as well which is automatically executed when the system is started. It is used to initialize the state of the

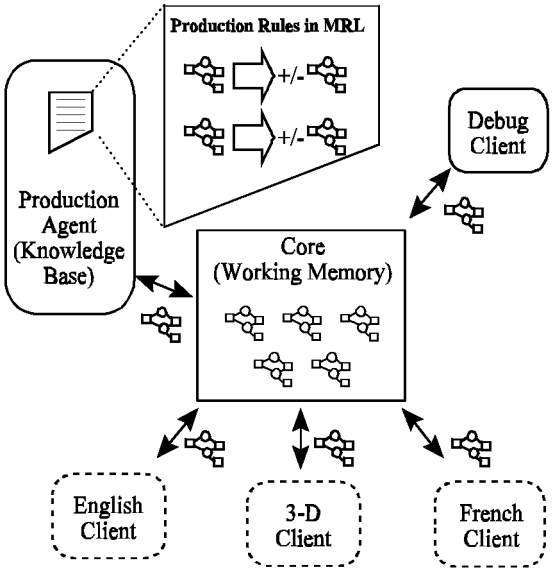

Fig. 2. Architecture of the Production System Layer

world by asserting a set of initial facts. The Mundo layer is unique for each world, although they may well share rules and facts in common.

Interaction with the virtual world is accomplished through clients that directly access the core of the Muto layer and assert facts about user actions. Any change in the Muto core is broadcast to the production agent, so it can respond, and to the clients so the world view can be updated for users.

4.2 The Muto Layer

MRL Rules A very important feature of the Ossa system is the Muto Rule Language which is used to describe a virtual world's model. The MRL uses a small set of keywords in combination with CG's expressed using CGIF to describe the production system rules. Typical production system rules are composed of a precedent and an antecedent. The precedent is a pattern which is matched against the working memory to determine if the rule fires. When a rule fires, the antecedent determines the changes made to the working memory. Any relationships between the facts matching the precedent and the changes made by the antecedent are usually described using labeled variables.

The use of variables adds considerable complexity when CG's are used to describe the rules and facts. It requires variables linking multiple graphs which lead to confusing rules. We solved this problem by combining both precedent and antecedent in a single graph. The graph describes the elements that must be matched, and what changes are made once the match is found. Thus, a single graph contains both the precedent and antecedent for a single rule.

The graphs themselves are called operational graphs since they incorporate all operations involving the graph into the graph itself. There are four basic operations associated with operational graphs: matching, asserting, retracting, and replacing. These operations are indicated by annotating the CG's with "plus" and "minus" signs to indicate assertion and retraction respectively. These annotations are added to nodes as CGIF node comments, so that any CGIF-compliant editor can be used to create and edit them. Unannotated nodes are considered to be *match nodes*. Replacement operations are specified by linking an *asserting node* and a *retracting node* with a coreference link.

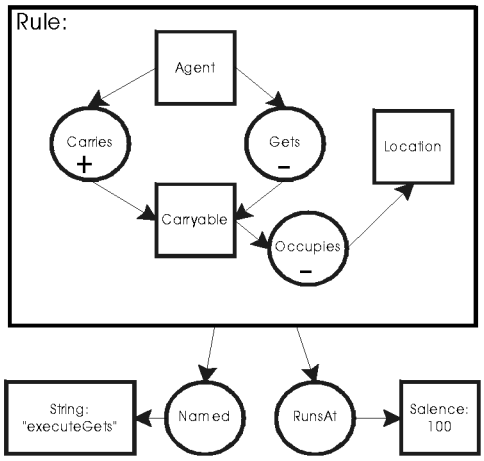

Fig. 3. A graphical view on an MRL rule

The diagram in Figure 3 shows a graphical view of an MRL rule which represents the action of picking up an object. The Rule concept is given both a name and salience (see below) using relations. The graph nested within the Rule concept is an operational graph with matching, asserting, and retracting nodes. The rule in Figure 3 and the rest of an associated MRL world file are shown in linear MRL notation in Algorithm 1[1].

Operational Graphs The basic way in which an operational graph is evaluated is as follows:

1. find all projections of the operational graph, O, in the working memory graph, M, ignoring any asserting nodes in O
2. if one or more projections exist, the operation graph is said to have fired
3. for each projection found for a fired graph

[1] Note that the CGIF syntax used in MRL is based on an early draft of the CGIF standard and does not conform completely to more recent versions of CGIF.

Algorithm 1 A sample MRL world file

```
world SimpleSample;
// A rule to initialize the world.
// It detects the presence of a [__Start] concept,
// retracts it, and creates some
// concept and relation types.
rule initializeWorld(salience 1000) {
  /* Match, and if found, retract a __Start concept. */
  if {{ [__Start;-] }} then {
    concepttype Agent;
    concepttype Gender;
    concepttype Location;
    concepttype Carryable;
    concepttype Male < Gender;
    concepttype Female < Gender;
    concepttype Brick < Carryable;
    concepttype Person < Agent;
    concepttype Man < Person, Male;
    relationtype Acts;
    relationtype Gets < Acts;
    relationtype Carries;
    relationtype Occupies;
  }
}
// Detects a "Gets" act, retracts the [Gets] concept,
// retracts the (Occupies)relation of the Carryable object
// and asserts the new (Carries) relationship
// between the Agent and the Carryable.
rule executeGets(salience 100) {
  execute {{ [Agent*x][Carryable*y] (Gets?x?y;-)
             [Location*z](Occupies?x?z) (Occupies?y?z;-)
             (Carries?x?y;+)
  }}
}
```

(a) any nodes in M that correspond to retracting nodes in O are removed from M
(b) any asserting nodes in O are copied and joined to M using matching nodes as join points

If multiple rules are matched by the current state of the working memory, conflict resolution is applied. There are many kinds of conflict resolution [16], but we have implemented an extremely simple form which establishes a precedence amongst rules by assigning them a *salience* value. Rules of higher salience are executed first. If two or more rules of the same salience are matched, they are all fired. For example, the rule shown in Figure 3 has been assigned a salience of 100.

MRL Statements The MRL contains four statements that control the effect of a rule.

An *execute* statement takes an operational graph as an argument. It simply evaluates the graph and fires it if appropriate. An *if-then-else* statement uses an operational graph as a condition. If the operational graph fires, the statement block following the *then* keyword is executed. If it does not fire (no projection is found), the *else* block is executed.

This allows for a rule to contain several, nested sections with different operational graphs. Graphs in different statement blocks can be linked via coreference labels. This forms an extremely powerful means for producing multi-function rules.

The last two statements, *concepttype* and *relationtype*, are used for declaring new types and can optionally specify sub- and supertypes for the new type.

Muto Clients The Muto system consists of the *core*, which holds the current state of the world, and a set of *clients* (or agents) that interact with the core to produce changes. The clients typically represent user interfaces which present some information about the world state to the user by querying the core and rendering the result. The rendering depends on how the client is designed to present the world. For a text-based client, the world may be presented in English text, or perhaps in some other natural language. Alternately, the client could provide a 3-dimensional rendering[2]. The clients also take user commands and translate them into operational graphs which are submitted to the core for possible execution. This clearly requires another level of translation depending on the type of client. This issue of client translation is clearly important, but is not addressed in this paper which concerns only the internal representation.

One special client, called the *production agent*, is responsible for executing the MRL rules that drive the world and respond to changes introduced by the user. This functionality is separated from the core to allow for several separate agents that would govern world control. This offers the possibility of future

[2] In Figure 1, the clients shown in dotted have not been implemented. Only the debug client, which uses annotated CGIF has been implemented at the time of writing.

modularity and allows for control schemes other than MRL. At present only a single production agent is used in practice.

The Muto Core All of the facts about the world are contained within the Muto *core*. In the current implementation, this is simply one, large (usually disconnected) CG represented using instances of the Notio API classes. The core is manipulated by clients issuing *core requests*. These are instructions for the core, but are called "requests" because the core may opt to ignore or only partially fulfill a request. This allows the core to prevent clients from causing problems or exceeding some fixed set of capabilities. There are four kinds of request:

1. *Query* Requests: The requests consist of a query graph. The results of matching this graph are returned to the issuing client only. Such graphs might be used to obtain information required for rendering a user's view.
2. *Change* Requests: These requests usually consist of an operational graph which is executed against the graph stored in the core. If the changes are accepted by the core, they are not immediately made, but are instead added to a *schedule* of changes. This prevents the results of a series of changes from interfering with each other. Commands from users and triggered events in the world typically result in change requests.
3. *Commit* Request: These requests simply ask the core to commit the changes currently pending in the schedule. For example, a client may issues several change requests that establish some effect they wish to have on the world. Once all have been submitted and accepted, the client would issue a commit request to activate the changes.
4. *Rollback* Request: These requests ask the core to clear all pending changes from the schedule. This can be used by a client to cancel a partially issued series of changes as the result of user input or the failure of a prior request.

Query and change request graphs are issued to the core using annotated CGIF. When changes are commited, the effects of the changes (i.e. the projections and asserted/retracted nodes) are broadcast to all clients via a Muto *event*. Clients can use these events to update their rendering of the world or as a basis for requesting new information. The production agent uses the events to detect changes in the world and test the appropriate rules in its world file to see if some response is required. Any response by the production agent is submitted in turn as a change request.

5 Conclusions

Ossa, as specified above, has been fully implemented and used to create a simple virtual world. This world offers features commonly found in MUD systems, such as taking and carrying objects and moving between different locations (a model of several locations at the University of Guelph campus). It also features some

more advanced abilities which are non-trivial, even in advanced MUD's, such as preventing entrance to a location. The world proved very easy to develop and involved considerably less code than equivalent object-oriented approaches (tens of lines vs. hundreds of lines). Making substantial changes to the behaviour of the world was easy as well, since they generally required the modification or addition of a very few rules, or simple changes to the type hierarchies. While not nearly as extensive as typical MUD's, this simple world has served as a successful proof-of-concept for the modelling approach and revealed some advantages and disadvantages to the approach.

Some care has to be taken while building in order to understand the interaction of the various rules. Unforeseen interactions can be a major problem in any production system. However, we think that the complexity gained through the interactions of simple rules can greatly enhance the resulting world. We also believe that tools can be developed to help designers detect and understand the interactions within a rule set so that they can be exploited rather than allowed to cause problems.

The current implementation offers only a single client for interacting with the system, called the debug client. The interaction is entirely via annotated CGIF statements that are used to issue commands and see the results. Natural language and graphical interfaces are clearly of great interest but a richer conceptual model is required before they can be attempted. In particular, spatial and temporal ontologies need to be explored that can be effectively rendered into different forms.

Some possible enhancements to the MRL and the Muto core include more sophisticated conflict resolution, more control over criteria used to find projections into the working memory, new operations that allow direct changes to the type or referent of a concept (this can already be achieved in a somewhat crude fashion using replacement), and improving the efficiency of the rule matching.

Conceptual modelling using CG's is an active area of research. Probably the greatest problem we encountered while working with Ossa was the temporal arrangement of events. In abandoning the procedural approach and adopting a production system model, we have to contend with the complex execution patterns that can occur. Guaranteeing that events occur in a sensible fashion is a task which we believe depends greatly on the modelling technique and ontology. We hope to exploit existing research on temporal modelling [17][18] using CG's in future efforts.

Finally, the Ossa system has served to demonstrate that VR may be built on the principled foundations of knowledge representation and logic. It is our hope that other researchers will apply advanced conceptual modelling technologies to VR systems and help to breach one of the most significant barriers to a complex virtual world.

6 Acknowledgements

Many thanks to Relu Patrascu and Rami Zeineh for their numerous suggestions regarding this work and also to the community at KobraMUD[3] for their input and guidance. Thanks to Stuart Statman for pointing in a promising direction.

References

1. Finnegan Southey, "Ossa: A Modelling System for Virtual Realities Based on Conceptual Graphs and Production Systems", MSc. thesis, supervisor Dr. James G. Linders, University of Guelph, Canada, 1998.
2. R.E. Fikes and N.J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving", Artificial Intelligence 2:189-208, 1971.
3. Richard Bartle, "Interactive Multi-User Computer Games", MUSE Ltd, British Telecom plc.,December 1990.
4. T Usaka, S. Yura, K. Fujimori, H. Mori, K. Sakamuram, "A multimedia MUD system for the digital museum", Proceedings. 3rd Asia Pacific Computer Human Interaction (Cat. No.98EX110), IEEE Comput. Soc, Los Alamitos, CA, USA,1998, xviii+474 pp. 32-37.
5. T. Meyer, D. Blair, D. B. Conner, "WAXweb: toward dynamic MOO-based VRML", 1995 Symposium on the Virtual Reality Modeling Language (VRML '95), page 105-108, ACM, New York, NY, USA, 1996.
6. Paul Tarau, Veronica Dahl, Stephen Rochefort, and Koen De Bosschere, "LogiMOO: a Multi-User Virtual World with Agents and Natural Language Programming", in S. Pemberton, editor, Proceedings of CHI'97, pages 323-324, March 1997.
7. Pavel Curtis and David A. Nichols, "MUDs Grow Up: Social Virtual Reality in the Real World",Third International Conference on Cyberspace, Xerox PARC, May 1993.
8. Sowa, John F., "Knowledge Representation: Logical, Philosophical, and Computational Foundations", Brooks/Cole, CA, 2000.
9. Jackson, Michael J., "Software Requirements and Specifications: a lexicon of practice, principles, and prejudices", Addison-Wesley, Great Britain, 1995.
10. W. R. Cyre, S. Balachandar, and A. Thakar, "Knowledge Visualization from Conceptual Structures" in "Conceptual Structures: Current Practices - 2nd International Conference on Conceptual Structures, ICCS '94", pp. 275-292, Springer-Verlag, Germany, 1994.
11. Petri, Carl Adam, "Kommunikation mit Automaten", Ph.D. dissertation, University of Bonn, English translation in technical report RADC-TR-65-377, Griffiss Air Force Base, 1966.
12. Kabbaj, Adil and Janta-Polczynski, "From PROLOG++ to PROLOG+CG: A CG Object-Oriented Logic Programming Language", Proceedings of the 8th International Conference on Conceptual Structures (ICCS 2000), Springer-Verlag, 2000.
13. Michael Chein, "The CORALI Project: From Conceptual Graphs to Conceptual Graphs via Labelled Graphs", in Proceedings of the Fifth International Conference on Conceptual Structures, ICCS'97, pages 65-77, Springer, 1997.
14. Rao, A. S. and Foo, N., "CONGRES: CONceptual Graph REasoning System", in Proceedings of the IEEE Conference on Applications of Artificial Intelligence, Orlando, Florida, pages 87-92, 1987.

[3] http://kobra.et.tudelft.nl

15. Southey, F. and Linders, J. G., “Notio - A Java API for developing CG tools”, in Proceedings of the 7th International Conference on Conceptual Structures (ICCS'99), Springer-Verlag, 1999.
16. J. McDermott and C. Forgy, "Production System Conflict Resolution Strategies", from Pattern Directed Inference Systems, edited by D. A . Waterman and F. Hayes-Roth, Academic Press, 1978.
17. John W. Esch, "Temporal Intervals", in "Conceptual Structures: Current Research and Practice", edited by Timothy E. Nagle, Janice A. Nagle, Laurie L. Gerholz and Peter W. Eklund, pp. 363-380, Ellis Horwood, 1992.
18. John W. Esch and Timothy E. Nagle, "Representing Temporal Intervals Using Conceptual Graphs" in "Proceedings of the Fifth Annual Workshop on Conceptual Structures", edited by Laurie Gerholz and Peter Eklund, pp. 43-52, Boston, USA and Stockholm, Sweden, 1990.