LOKI: A Logic Oriented Approach to
Data and Knowledge Bases Supporting
Natural Language Interaction

Jean-Louis Binot, Bart Demoen, Karl-Heinz Hanne
Levy Solomon, Yannis Vassiliou,
Walter von Hahn, Tom Wachtel

August 1988

FORTH/CSI/TR/1988/015
Technical Report Series
Computer Science Institute, FORTH
P.O.Box 1385, Heraklio, Crete, 711-10 Greece
Tel.: +30.81.229324, Fax: +30.81.229342, Telex: 262389 CCI GR
E-mail – UUCP: mcvax!ariadne!yannis

† "Foundation of Research and Technology – Hellas," formerly "Research Center of Crete" (RCC).
LOKI: A Logic Oriented Approach to Data and Knowledge Bases supporting Natural Language Interaction

Jean-Louis Binot (BIM, Belgium), Bart Demoen (KUL, Belgium), Karl-Heinz Hanne (Fraunhofer-IAO, Germany), Levy Solomon (SCS, Germany), Yannis Vassiliou (CCI, Crete), Walther von Hahn (U. Hamburg, Germany), Tom Wachtel (Scicon Ltd., UK).

Abstract

The aim of the LOKI project was to provide a set of advanced and performing prototype tools based on Prolog for improving the support of, and the access to, data and knowledge bases. The project has been completed in July 88. This paper presents its main achievements, which concern the fields of knowledge representation, knowledge base support, and natural language and graphical interaction with data and/or knowledge systems.

1. Project description

LOKI (Esprit project 107) stands for "a Logic Oriented Approach to Data and Knowledge Bases supporting Natural Language Interaction." This project, which has been performed jointly by a consortium of partners from Britain, Belgium, West-Germany and Greece, has attempted to integrate under a common theme research and development work relating to different fields. The natural language work has been done by Scicon Ltd. (B. Inlaha and T. Wachtel) in London, the University of Hamburg (W. von Hahn, H. Horacek, C. Pyka, M. Schroeder and M. Tarnowsky), the Fraunhofer-IAO and the University of Stuttgart (J. Hoepelman and P. Wetzel) in Stuttgart and BIM (J.-L. Binot, L. Debille and I. de Zegher) in Brussels. The work on graphics has been performed by the Fraunhofer-IAO and the University of Stuttgart (K.-H. Hanne, B. Arndt, A. Graebel, T. Manz and O. Boehm). The work on knowledge representation and the knowledge support system has been performed by ECS (L. Solomon, J. Gallagher, R. Meyer and G. Ulrich) and the CCI (Y. Vassiliou, J. Mylopoulos, A. Borgida, T. Tsaladou, M. Kouronakis and M. Marakakis) in Crete. Finally work on the Prolog environment itself has been done at BIM (B. Demoen, R. Venken).

The aim of the LOKI project was to provide a set of advanced and performing prototype tools based on Prolog for improving the support of, and the access to, data and knowledge bases. To that end, specific subgoals have been identified:

1. Special emphasis has been put on natural interaction with data or knowledge systems. Two ways of creating natural interaction have been explored: a natural language interface and a direct manipulation interface. The natural language work has concentrated on developing a prototype for querying (relational) databases; however, the results presented below can be generalized and built upon to access other knowledge based kinds of interaction.
2. Simultaneously, great attention has been paid to the problem of knowledge representation and use. Natural language understanding, as it is well known, is a knowledge-based process. Coupling a knowledge-based component to a relational database is also a most logical step to take in the Prolog paradigm. Work has concentrated here on the development of a knowledge representation language called CML (Conceptual Modeling Language) and to the development of a knowledge support system able to provide both a convenient user interface for knowledge editing and a powerful inference engine.

3. To support the development of the above tools in a Prolog environment, numerous extensions to Prolog itself have been developed, such as an interface for relational databases, an integration with graphics and windowing and an interprocess communication tool.

The different tools developed are all parts of an integrated support system for a data or knowledge base. A general (and somewhat simplified) view of this system, which shows at the same time the architecture of the whole project, is shown in Fig. 1, below.

**FIGURE 1: LOKI ARCHITECTURE**

At the core of the system is the CML knowledge representation language, and more specifically the CML kernel, which consists in predicates for creating and deleting CML assertions from the CML knowledge base. The figure shows only access paths between modules. Additional control paths exist between pairs of modules, such as between NL and communication layer, NL and CML support system, CML support system and communication layer.

The CML language is the uniform medium of communication between the different parts of the system. Thus, the natural language interface, for example, interacts with CML not only for accessing its "world model" (the knowledge base about the world in general and about the domain in particular, which is expressed in CML) [Imlh87] but also for accessing the database. Indeed, input natural language queries are translated into CML expressions, which...
are then "proved" in the CML environment by using the CML support system. The database being viewed as a simple extension of the knowledge base supporting the proof. The implementation of the CML support system is also based on the CML kernel: the support system provides a CML inference engine and a menu driven user interface between the user and the knowledge base. The CML support system may also be activated from the natural language interface. Finally, the direct manipulation interface interacts also with the rest of the system through CML, via a communication module.

The prototype developed has used as testbed a database called PRADOS about project management. Porting to another database (SEMINAR) has started during the last months of the projects. Other experiences of porting have been done with great success, although on a smaller scale, by some partners: these experiences indicate that the overall portability of the software developed with respect to database domains is very high.

The rest of the paper describes separately the achievements obtained in each aspect of the project.

2. CML: A Conceptual Modeling Language

2.1. What is CML?

CML is a knowledge representation language that has been developed in the LOKI project for modeling the domain of discourse and the requirements of information systems. It provides a number of novel features and incorporates many ideas into one framework. CML is based on an object-oriented framework, with many improvements such as organization principles, exception handling mechanism, a sublanguage for talking about the universe of discourse, an assertion language for constraints on KB objects and the integration of time.

CML provides also linguistic facilities for describing a knowledge base, the application domain and relationships between the two. As in any object-oriented framework, CML is based on the idea of organizing a model of the (or of a) world around concepts (objects) and the relationships existing between them. Moreover, similar concepts are grouped in classes which are themselves concepts and are likewise grouped into even more abstract classes (meta-classes). The object-oriented framework adopted by CML is based on semantic network ideas and has evolved out of research done for the TAVIS [Mylopoulos and Greenspan 83] and RML [Greenspan 84] languages. The abstraction principles used by CML are:

- classification: facilitates the grouping of objects having common properties into classes.
- generalization: classes can be organized into a hierarchy from most general to most specialized.
- aggregation: it facilitates viewing an object as a collection of its properties.

Assertions which impose constraints on the possible relationships are also encoded through properties. Property categories offer an other way to impose constraints on the values of the properties.

CML enhances the above framework with the following ideas:

1. properties/links are treated as first class citizens and enjoy all the privileges of objects; this provides among other things facilities to define metaproperties;
2. all units of a knowledge base are assumed to have a time inferred to as part of their internal structure. The justification of this is that requirements models should provide a history of
the domain [Stanley 86], thus emphasizing the importance of capturing the dynamics of the application in a declarative fashion by explicitly acknowledging the passing of time [Allen 84]. According to this, what is being modeled is not the application domain, but the knowledge about the domain.

3. assertions about the knowledge base or the domain are also treated as first class citizens (objects) and constitute an integral part of a knowledge base.

4. talking about the referents of the knowledge base objects opens the door to statements about the accuracy/completeness of informations in the KB

5. the KB itself is treated as an abstract data type.

6. In many practical situations it is natural to overabstract by ignoring special cases during early stages of modeling, which lead to the need of overriding previous generalizations or of going back to modify the generalization assumptions. CML provides an exception handling mechanism which allows constraints on the structure of the knowledge base to be cancelled within a particular context.

2.2 The knowledge level interface of CML.

The view of the Knowledge Base (KB) as an abstract data type is one of the basic ideas of CML. A KB can be completely specified functionally with no regard to how it is implemented [Levesque and Brachman 84]. Essentially, a knowledge base is treated as an abstract data type that interacts with a user or system through a small set of operations. This approach focuses not on the "symbol level" of data structures and property inheritance, but on the "knowledge level": on what the system knows about the world in which it is embedded.

In CML the knowledge level interface is primarily based on two operations: TELL and ASK, while other high level operations such as RETELL, UNTELL and RETRIEVE are being added. The meaning of these operations is the following:

a. ASK accesses knowledge which is known:

\[ \text{ASK : Knowledge } \times \text{Query } \rightarrow \text{Answer} \]

b. TELL gives us the ability to augment what is known:

\[ \text{TELL : Knowledge } \times \text{Declaration } \rightarrow \text{Knowledge} \]

The implementation of the TELL operation consists of a parser, that parses a CML declaration, a semantic checker which checks if certain semantic network postulates are satisfied by the information told and finally a consistency checker which proves the integrity of the knowledge base. The implementation of ASK is divided in two parts, the implementation of a query processor and a temporal reasoner.

2.3 Modeling of time and temporal reasoning in CML.

CML offer facilities for representing temporal knowledge and reason with it. Every CML object has a temporal component, as already mentioned. The time representation provided by CML is interval based, following [Allen 84]. The temporal relations supported are axiomatized in terms of the primitive "meets" defined in [Stanley 86]. Temporal relationships between objects are encoded as:

\[ p1 \text{ THETA } p2 \text{ iff } \text{when}(p1) \text{ THETA when}(p2) \]
where p1,p2 are CML objects, "theta" any temporal relationship and "when" the function which returns the time interval attached to each object.

CML extends Allen's theory by modeling elastic time intervals (open ended), and time constants for binding a sequence of actions with the real world timing system (conventional time and date). Time intervals and their relationships (e.g. before, after, during, meets, ...) are represented by a graph where the intervals are nodes and the relations edges.

The CML temporal reasoner provides the following services:

1. extraction and storage of temporal knowledge from user assertions, as well as computation and storage of the knowledge obtained by combining existing and incoming temporal information.

2. answer to temporal queries of the form "t1 THETA t2", where t1, t2 are temporal intervals and THETA any temporal relation (e.g. before, during, costs,...).

The intractability of known algorithms dealing with time intervals has directed us to translate the interval based representation to an equivalent point based one. The algorithm for temporal reasoning we are using is based on the work of [Villain and Kautz 86], modified for the CML's needs; this algorithm has been proved to be sound and complete.

2.4 The CML Support System

During main phase 1 of the project a CML Support System was designed and implemented [Gallagher and Solomon 86]. The implementation, which covers most of the important aspects of the theoretical description of the CML language [CCl and SCS 86], [CCl 87], was further developed in main phase 2 and consists now of a set of basic predicates called CML Kernel, a set of reasoning tools and a user menu [Meyer and Solomon 88]. The CML Kernel + reasoning tools provide an interface for the programmer, whereas the interaction with the user is supported by the menu.

2.4.1 The CML Kernel

The CML Kernel consists of predicates for creating, deleting and retrieving information from the CML knowledge base. The query predicates support inheritance. The kernel is designed to be implementation independent since it hides the internal form of the stored information and provides the programmer with a well defined interaction level to the functionality of the CML system.

2.4.2 CML Assertions

A CML model is translated to a set of assertions which may be regarded as a logical theory [Gallagher and Solomon 87]. This logical theory consists of two parts.

1. A set of general clauses (KB) of the form:

   \[ P1 \ AND \ P2 \ AND \ \ldots \ AND \ Pn \rightarrow Q \]

   where n \geq 0 and each Pi is a positive or negative literal (or set construct). If n=0 the assertion is a fact, otherwise a rule. The knowledge base (KB) represents the "state of the world" and non-negated atomic facts describing property values may be derived from it.
3. LOQUI: a Natural Language Interface

LOKI has produced a natural language interface for relational databases called LOQUI, which integrates in a practical system some of the most recent theories and techniques in Natural Language Processing. LOQUI is a multilingual and portable system, offering currently interaction in English and in German and easily adaptable to new application domains. From a linguistic point of view LOQUI features a reasonably extensive grammatical coverage (questions, statements, orders, sentence fragments, relative clauses, active and passive sentences, negations, coordinations, etc.), complete morphological processing, spelling correction, and English and German generation. One of its main assets is a sophisticated discourse manager able to deal with phenomena such as pronoun references, ellipses, cooperative answers and presupposition detection and correction. The architecture of LOQUI is shown in figure 2. A more complete general description may be found in [Debille and Binot 88].

![Architecture of the LOQUI Interface](image)

**FIGURE 2: ARCHITECTURE OF THE LOQUI INTERFACE**

One of the most important aspects of LOQUI is that it is intended to be more than just a question-and-answer system. It is being designed so as to have a rich dialogue capability that will allow conversations to proceed with the coherent fluidity that is characteristic of human interaction, thereby allowing both system and user the possibility of exploiting contextual information in order to minimize constraints on expressiveness. In short, LOQUI operates relative to the context that has been built up during a conversation, rather than returning to a neutral state after each exchange.

LOQUI accepts input in English or German. The output of the parser is a logical form that is mapped into a database query. The results are displayed to the user in natural language. The system is able to deal with substantially corrupt input, including spelling errors, and errors stemming from user misconceptions.

No self-respecting natural language interface can operate without a world model. In LOQUI, this is written in CML see [Bibel et al. 1986], [Inlah 1987], [Gallagher and Solomon 1986],
The world model is represented as a hierarchy of classes with instanced and properties. In addition, rules may be defined on classes to express knowledge that can be used in inferencing, and elsewhere. For instance, rules of this type are used to relate the world model to the back-end in order to get the correct database mappings form semantic representations. Moreover, these rules, as well as other aspects of the world model, are used in error recovery, in particular to make hypotheses about the user's intended meaning in the face of apparent semantic deviance or the impossibility of finding a reasonable mapping to database access formalisms. Examples of such error recovery will be given below.

Error recovery is an important part of discourse management, and spelling correction is an important part of error recovery. In conjunction with the ability to recover from semantic deviances and the ability to generate strings from semantic representations, it provides a useful tool in allowing a dialogue to continue in spite of errors, but at the same time alerting the user to the corrections that LOQUI has effected. Examples of this will be given below.

The discourse manager is invoked after every user input has been evaluated, and after every response by LOQUI. It is a bottom-up dialogue parser. That is, it takes each input, and, given a current dialogue state, decides how to produce a new dialogue state that takes into account the current input. It also establishes a set of expectations as to what the next move might be, thus laying the foundation for conversational repair in the face of uncooperative user behavior. A dialogue state consists of a hierarchical structure of move labels which refer to the dialogue memory -- a permanent record of the semantic representations of all the moves in the current session, together with other information about each move.

LOQUI's dialogue memory is more than just a list of inputs. Relationships between moves are recognized, and the whole dialogue is represented as a tree structure that reflects these relationships. The discourse manager can also identify subdialogues, where the topic of the exchange is the exchange itself, with a pop back to the matrix topic in due course and can also introduce hypothetical moves which are a reflex of the user failing to respond as expected. Further details are available in [Wachtel 86]. The discourse manager is important for LOQUI in order to establish a context of discourse to which reference can be made in subsequent parts of the dialogue.

At any point during a conversation with LOQUI, the system is aware of a set of discourse referents. These are CML objects whose existence as discourse referents has been sanctioned in one way or another. The list of discourse referents is maintained in the following way. A CML object is sanctioned as a discourse referent either if it is mentioned by the user or by LOQUI, or was referred to indirectly during the evaluation of a user input. Discourse referents are not restricted to objects such as people or projects, but include relations, and other more abstract objects.

Discourse referents are ephemeral objects in that they are created anew each time the discourse manager is called. However, since reference resolution picks up objects from among the discourse referents, they do get carried along through the conversation -- but only as long as they are still being referred to somehow. That is, a discourse referent is like a ball being tossed between the discourse manager and the reference resolution component, and it is a discourse referent for as long as neither side drops it.

However, there is always one discourse referent that is salient, and that is the topic of the current topic is. Topics are more stable than other discourse referents, in that they do not cease to be discourse referents if they are not referred to in the subsequent move. In order for an object to cease to be the topic, there must be sufficient grounds for the discourse manager to decide that there is a new current topic which has ousted the old one. In other words, dis-
course referents die through neglect, unless they are the topic, which has to be killed by another topic.

The following dialogue illustrates the interaction between reference resolution and the topic of the exchange, and how decisions made about reference resolution affect the course of the dialogue. It also shows LOQUI's cooperativeness in its replies, and the interpretation of a null input as a request for elaboration.

*** does CP work on anything

Yes, Claudius Pyka works on LOQUI.

*** who leads it

Walther von Hahn.

*** what describes the project he leads

Well, it could be LOQUI-Report NLI 4711: Final Report, HAM-ANS Final Report, HAM-RPM Final or Report on LOQUI2. There is more than one such project on the database.

*** (empty user input)

Walther Von Hahn heads the project LOQUI2. LOQUI2 is described by Report On LOQUI2. The HAM-RPM task is headed by Walther Von Hahn. HAM-RPM is described by HAM-RPM Final Report. Walther Von Hahn is the head of HAM-ANS. The task HAM-ANS is described by HAM-ANS Final Report. The task LOQUI is led by Walther Von Hahn. LOQUI described by LOQUI-Report NLI 4711: Final Report.

*** does jm work on a project that vh's boss leads

Yes. Walther Von Hahn reports to Nick Ostler. LOKI is lead by Nick Ostler. Jim Madar works on LOKI.

*** does jg lead alvey

No. John Gallagher leads SCS-CML.

LOQUI's ability to resolve reference is not constrained by the length of the dialogue. That is, there is no constraint that the referent should have been introduced at most two moves earlier, or anything of that nature. LOQUI always operates with a set of current discourse referents, and this current set is amended as the dialogue proceeds.

In cases where the user's input is well-specified semantically, but contains spelling errors, or semantic deviance, or both, the world model and/or the spelling corrector is invoked in order to attempt to make sense of the user's input. In addition, the generator is used to clarify to the user what corrective action has been effected.

*** which dog describes the cat that NO leads

I assume you mean "Which document describes the activity that is headed by NO?" rather than "Which dog describes the cat that NO heads?"
LOKI Report 111: Results

*** which old bitch runs LOKI

I assume you mean "Who is LOKI headed by?" rather than "Which old female dog heads LOKI?".

Nick Ostler

*** which hippy chat runs the LOKI gog

"which hippy chat runs the LOKI gog"...? I assume you mean "Who is the activity headed by?" rather than "Which happy cat heads LOKI?".

Nick Ostler

Where possible, LOQUI attempts to maintain the dialogue by providing at least the information requested, but also pointing our any detected presupposition failure. At present, this is restricted to existential and uniqueness presupposition failures.

*** which document describes the project that precedes the project that the person that VH reports to leads

Well, it could be Ham-RPM Final Report or HAM-ANS Final Report. There is more than one such project on the database.

*** (empty user input)

LOKI follows the task HAM-ANS. HAM-ANS Final Report describes HAM-ANS. Walther Von Hahn reports to Nick Ostler. Nick Ostler is the leader of LOKI. LOKI is a successor of HAM-RPM. HAM-RPM Final Report describes the HAM-RPM activity.

*** who works on the project that Paul Bates leads

There is no such project on the database.

4. Direct (graphical) manipulative interface

The most prominent medium in human communication is natural language (NL). NL is always used in a certain situation - at a certain time and a certain place by people (or systems) who share a great deal of both, situational perception and general knowledge. These boundaries determine the comprehension of natural language.

Direct Manipulation [Shneiderman '82] on the other hand provides another way of natural man computer interaction.

Deictic reference is one effect of the combination of DM and NL and is a crucial point in man computer interaction. Traditional categories of deixis are person, place and time [Hoepelman et al. '86]. In screen oriented interactions, place deixis - in the first approach to a pixel on a screen - is of prominent relevance. In human communication, deixis is oriented in an egocentric way, i.e. both (or all) 'speakers' and 'listeners' have to share a common model of the
viewpoints of the different participants. In the context of this project, the emphasis of evaluation of deictic expressions is put on screen-oriented acts of selection in combination with NL dialogues. This inclusion of deictic actions in the user interface leads to the concept of combined multi-modal computer interface, which tries to share the advantages of generic communication modes but to avoid their disadvantages [Hanne and Hoeselman 88].

Examples of the power of NL/DM interaction include the possibility of ‘talk’ about objects the user has no name for, or to use simple pointing, e.g. on a map, instead of a complicated spatial description. Applications on maps can help to talk about temporal aspects using spatial relations, if there is an implicit or explicit time axis. Objects which are already synthetic, e.g. business graphics, are well suited for combined NL-deictic interaction: since they are created by the system, an internal representation is implied.

At the end of the project, first results of the integration of deixis handling with the NL interface will be obtained. However, an important part of the work on graphics in LOKI has also been devoted to the development of suitable tools for the generation and the management of graphical depictions, which are necessary for dealing with deixis in any useful way. These tools are described below.

4.1. Integrated Prolog Graphics DM Interface System

The integrated Prolog direct manipulative interface system [Hanne and Graeble 87] was aimed at the design and implementation of direct manipulative user interfaces, as well as at the development of a communication layer separating the different programming paradigms (logic oriented declarative in the application / procedure oriented sequential in the user interfaces / object oriented in the definition of graphical screen objects). Tools for the generation, the management of graphical objects and the interaction with the other LOKI components have been added. A suitable knowledge representation formalism (CML) can be used, interpreted and adjusted to the actual interface. All these provide safe communication and synchronization based on UNIX and tailored for Sun workstation with bit-map display, and the available graphic- (and windowing-) primitives.

4.1.1. Interface-Application Communication layer

A major problem in the implementation of interactive software is the internal communication, synchronization, and compatibility of different programs. Therefore, a Prolog-C binding mechanism based on UNIX pipes forms the basis of the implementation of direct manipulative interfaces. The design is based on the following ideas.

The internal structure of programs running on window systems (e.g. SunTools, XWindows) influences the concepts required for direct manipulative interfaces. Window systems in general provide a uniform centralized I/O mechanism. In addition, this mechanism handles screen updates, like window movement, zooming, hide, expose events, etc.

Similarly, direct manipulative/NL interfaces should not be controlled exclusively by the application system. A simple example is the window resizing control of a graphics output module. A solution can be seen in separate coexisting programs. Each program is a complete unit which might be run separately, and does not need connections to other programs. Both the implementation and debugging of these separate programs are made easier by avoid side effects from other tasks, since their only usable communication path to the 'outer world' is a well tested interface with included debugging aids.
4.2 Interactive graphic interface

4.2.1 Interactive business graphics in CML

A module for the creation and interactive manipulation of business graphics (PieBar, Pie charts and Barcharts) has been developed. The CML part of the business module consists of the four classes "Window", "Graphic", "Text" and "Selections" and their instances. The classes Window, Graphic and Text have corresponding classes in the graphical parts of the PieBar module. The special class 'Selections' is defined in order to allow a simple access to the CML representation of displayed and user - or application - selected objects.

If the CML representations are modified, the corresponding displayed graphics objects are modified accordingly and vice-versa. The class 'Graphic', for instance, describes the semantics of single bar or pie segments using properties like:

```
selectable, selected, selected label, font, size, value,
position x, position y, position z, layout, frame, style bold,
style underline, shadow.
```

The CML representation is changed in parallel to the Prolog processes using the communication layer. This implies that a programmer or Knowledge Engineer has nothing to know about specific graphic properties or about the implementation.

4.2.2 Form oriented user interface

A graphic component for form oriented user interfaces allows dynamic creation, deletion and control of forms on the display. The application has not to know anything about the hardware and software limits of the machine used, nor about drawing algorithms, event handling and other interface specific tasks. The Language Independent Form Tool (LIFT) is adapted to a concept model of CML. Every change in the world of the CML objects is directly reflected by a change in the "display" world of form objects like frames, buttons, texts, menus or sliders. The interface tool allows simple construction and use of hardware independent user dialogues based on forms. User entries can be checked automatically and actions can be limited to events like selecting objects, entering texts or selecting in menus. These actions allow self-modifications of the interface, reactions to the user (including help) or messages to be sent by the application. Using CML a full user interface can be built: change to the CML objects are reflected by modifications of the objects of LIFT and changes of LIFT objects by the user are reflected by modifications of the CML objects. The process-to-process communication between LIFT and Prolog running the CML support system is done by the communication layer.

4.2.3. Graphic shell

The Graphic Shell (GS) is a tool to create graphical surfaces for (mainly) knowledge based systems. The communication between the GS and the application program is based on the communication layer. Therefore the tool and the program using it are running as independent processes. The GS has the following characteristics:

- The graphical knowledge represented in the GS is frame (object) oriented;
- Graphical commands generate an internal representation of the displayed objects;
- Graphical classes and instances are supported (instance hierarchy). Between frames exist not only part-of links but isa links. Inheritance of graphical attributes through both kinds of links is possible;
• Prolog coupling: the generation of graphical objects and the display of the hierarchy is possible with Prolog predicates via the communication layer. Couplings to other languages are possible too.

• CML coupling: the internal GS frame structure is represented in CML. This implies that changing a slot value of a graphical instance changes the corresponding value displayed on the graphical surface.

• Frame editor (Object editor): besides Prolog commands or CML modifications, it is possible to generate graphical frames with the help of a graphic editor.

• Pointing: using the mouse, the user can not only select a single object but also combined objects. This allows the user to point to the whole object as well as to its parts.

5. The Prolog level

In 1983, almost at the same moment main phase I of LOKI started, a joint project of BIM and K.U.Leuven, sponsored by the Belgian Government (DPWB-SPPS), started: this project aimed at implementing a reliable and fast Prolog system, which would be commercialized by BIM. The Prolog system developed is now known as BIM-Prolog. Since BIM was the prime contractor for LOKI and since the software to be developed in LOKI was to be written in Prolog, it was a natural decision to use BIM-Prolog in the LOKI project and to use LOKI as a test bed for the implementation of BIM-Prolog. The Prolog programs developed in LOKI would make heavy demands on BIM-Prolog: speed as well as reliability would be tested to the extreme, especially by the natural language interface. BIM also hoped that the functionality and user interface of built-in predicates and the debugging facilities would be influenced by the comments of the users of BIM-Prolog inside LOKI.

LOKI produced many bug reports which resulted in a more reliable BIM-Prolog system on the whole. Also, since the programs developed in LOQUI are large - now up to 20000 lines of Prolog code in about 50 files - and use a lot of atoms (it is a natural language system !), the garbage collectors of the code space and the constant table were tested continuously. Moreover, debugging of Prolog programs, mode declarations, the module system and the built-in predicates improved from the feedback of the partners. New facilities were introduced in the BIM-Prolog system, like erman/lemore flexibility in choosing the size of a BIM-Prolog system (LOQUI is large and always growing, some tools remained small), retract of static code (the CML support system relies on it for its efficiency), interrupt handling (needed in the communication layer of the direct manipulation interface), compaction of compiled code (some partners work with very small disks !), just to mention a few.

Worth mentioning apart is the use of the external language interface of BIM-Prolog in the project: at first this interface was only used for the coupling with an external database (Unify in this case), but later it became clear that some very procedural parts of LOQUI could better be written in C, so that now LOQUI relies on this external language interface for critical parts of its code like spelling correction and most of all the lexical database: the design of 'backtrackable C-functions' has been influenced by considerations in LOKI. On the whole, BIM-Prolog has benefited a lot from being used in the LOKI project as the main implementation language.

6. Conclusions and industrial perspectives

A prototype of the full LOKI system, including a CML support system, the LOQUI natural language interface and the direct manipulation interface, has been realized. The main steps toward the integration of these components have been done, and the first results of that integration obtained. Thus the NL interface can activate the CML support system, and use the
CML prover to solve queries. The graphical component is also integrated with CML and can be used to display CML objects. Finally, first results of the integration between NL and direct manipulation will be obtained at the end of the project. Of course, it is certain that more work would be needed to exploit fully the power of the resulting integrated system. However these first results suffice to show that LOKI is a successful example of development of a set of tools increasing the power of the Prolog/database/knowledge base paradigm in an integrated way.

A major industrial interest of this project is provided by the emphasis put on very high level interfaces. The importance of interfaces, the only part of a system really in contact with the user, cannot be underestimated. Interface technology itself is in full evolution. The generalization of the use of powerful workstations with bitmap screens and mouse, the considerable importance taken by graphics, the emergence of ergonomic considerations, and the progress in natural language processing have contributed to the transformation of interface technology into one of the key strategic areas in information processing technology.

By providing the stimulation for the full integration of all graphical possibilities of SUN workstations with BIM Prolog, and by providing also results and tools to build well designed interfaces with graphics, the project has provided one of the key basic technologies which will permit the development of real industrial applications in PROLOG (knowledge based, but also software engineering and other more conventional applications).

Another very significant industrial impact comes from the results obtained in natural language processing. Natural language currently represents only a small market, but the technology is slowly maturing, and is considered as a key technology for the coming decades, especially considering the slow but constant and significant progress in the area of speech recognition and talkwriters. As the Ovum report on natural language processing (NLP) noted, the increasing size and complexity of databases and other software require more flexible "wide bandwidth" interfaces for communication with their users, and NLP is one way of meeting these new demands. [Wasserman 87] evaluated the global NLP market at about 110 millions dollars in 86, and estimated that it would increase to 440 millions dollars for 1990. The European portion of such a market is estimated at approximately 17%. It should also be noted that so far the field of natural language processing has been largely dominated, from the technical point of view, by the Americans.

Natural language processing is a technology with a slow maturation rate: it takes time and effort to be present and reach a good technological level. For this reason, it is a technological field that small and medium companies can only penetrate with the greatest difficulties, if at all. The ESPRIT project LOKI has provided for the industrial partners of this consortium the opportunity of making this penetration effort, which resulted not only in a prototype of a natural language interface for relational databases, but also in invaluable experience and knowhow in the field. The prototype LOQUI has reached a sufficient level to serve in the relatively short term as the basis of a good NLP interface product; exploratory missions in the USA leads us to believe that this prototype is currently on a par level with the best American efforts in the domain, and that given its multilingual capabilities, it can become a leading product in the European market, and even possibly worldwide.

More efforts, of course, should be devoted to the continuation of the lines of research defined in this project. The natural language work can be completed by interfaces for other European languages, such as French and Dutch. The linguistic scope and robustness of this interface can also be extended. The work on the integration of knowledge representation, natural language and graphics can be seen as a first step towards "multimode interfaces". Implementations of conceptual modeling systems can certainly be made more efficient, and the expressive
power of such system can still be extended. These are some of the directions where LOKI's results can and will benefit to future research.

REFERENCES


Cretan Computer Institut (CCI) Greece, SCS Technische Automation und Systeme GmbH West Germany, "Conceptual Modelling Work Package Progress Report on Tasks CM5, CM6, CM7", July 86


M. Stanley, CML: "A Knowledge Representation Language with Application to Requirements


Francis Wasserman, director of the "Bureau d'Informations et de Previsions economiques francais", market estimates published in "OI Informatique, Mai 1987.

Wachtel, T. "Pragmatic sensitivity in NL interfaces". Proceedings of COLING 86.
2. A set of integrity constraints (IC) of the form:

\[ P_1 \text{ AND } P_2 \text{ AND } ... \text{ AND } P_n \rightarrow \text{fail} \]

where each \( P_i \) is a positive or negative literal (or set construct) and the proposition fail is false in all interpretations. The set of IC is used to express constraints on all states of the model. This may be interpreted as negative information about the premises:

\[ \text{not } P_1 \text{ or not } P_2 \text{ ... or not } P_n \]

Proofs of negative facts (in the given assertion) rely on negation by failure for certain predicates e.g. "built-in" and for others the completion is explicitly defined by a predicate negation(...).

2.4.3 Consistency Checking

The discovery of inconsistency amounts to proving \( P \text{ and not } P \) or equivalently showing that no model satisfies the theory. This is of course not possible with a prover based on negation by failure.

The support system contains the Consistency Check of a class, where all the constraints of a class are checked. The class is considered to be closed (which means that the closed world assumption applies to it) and for every constraint in the class we try to prove the premises. Thus we have proven a contradiction because we proved the premises and also its negation (fail is false in all interpretations). This method is appropriate in order to prove general inconsistency for the theory KB+IC when no additional information is given.

Supposing we start with a consistent theory (at the beginning no facts are in the database) it is possible to use a more efficient method to detect inconsistency. When a new fact \( P \) is added to the KB in the CML model we try to prove that not \( P \) is derivable from KB + IC which is equivalent to the prove that KB + IC + P is inconsistent. A detailed discussion and a proof of this claim can be found in [Gallagher 88].

The same method can be used in order to derive negative information by using the contrapositive form of the assertions in KB+IC:

\[ P_1 \text{ AND } P_2 \text{ AND } ... \text{ AND } P_n \rightarrow \text{not } P_i \]

2.4.4 The CML Prover

CML rules are defined by the assertion language definition as specified with DCG [Gallagher and Solomon 86]. It allows instance and property/value expressions as well as built-in calls and set expressions. The assertion language can be used as a query language within the prove predicate, which accepts a list of literals to be proven as its argument:

\[
\text{prove} (\text{ASS}) \quad \text{If the assertion ASS is provable from the knowledge base, prove succeeds (instantiates free variables).}
\]

Queries concerning the property values of CML objects are answered by instantiating the "value" variables in a given assertion ASS.