

ISRN SICS-T--91/10--SE

Aspects and Experiences of MIS and SICStus Prolog

by
Nils Hagner and Ingvar Olsson

Aspects and Experiences of MIS and SICStus Prolog

Nils Hagner	Ingvar Olsson
Infologics AB	Infologics AB
Kungsgatan 62	Aniaraplatsen 6
P.O. Box 1713	P.O. Box 91
751 47 Uppsala	191 22 Sollentuna
SWEDEN	SWEDEN
Phone: +46 18 18 50 00	Phone: +46 8 92 20 00
Fax: +46 18 18 52 58	Fax: +46 8 96 08 46

August 15, 1991

T 91:10

Abstract

The ISP version of SICStus Prolog is an advanced Prolog system which can be used in a large variety of applications. Prolog, however, has not been very widely used in data-intensive MIS, management information systems, applications which are centered around one or more databases.

In this document, a (possible) relationship between MIS applications and SICStus Prolog is investigated. In the course of the investigation, a sample application has been developed. It is a running prototype of a decision support system for fault analysis in digital (AXE) switching systems.

A major point in the document is a set of "guidelines" for SICStus Prolog and MIS applications. These points describe a few hints on when SICStus Prolog can or should be used in MIS applications, as well as when SICStus Prolog should not be used.

1 Introduction

The aim of the ISP project has been to develop a version of SICStus Prolog — a Prolog system [Col73, Kow74] specially tailored to meet the demands from the industry. Among the demands are issues of integration, user interfaces, and database management. As a part of the ISP project, a sample application was built, using many of the built-in facilities of ISP. The aim of that subproject was to evaluate the usefulness of SICStus Prolog in data-intensive MIS applications centered around a database, and to evaluate some novel techniques for elevating such applications.

The subproject has produced various results. Perhaps the most important result is the identification of certain pitfalls that should be avoided and certain techniques that should be used. Our hope is that this result, presented here as a set of "guidelines," provide some advice both for those who consider using SICStus Prolog in MIS applications, and to those who continue the work on maintenance and development of SICStus Prolog.

The application in question concerns a fault analysis database used by the Swedish Telecom. This database contains various information about reported and corrected (software) faults in AXE systems. The application developed is used to find the appropriate information about faults when analysing new faults. The application development has been aimed at a stand-alone prototype system. The principles and techniques used, however, can also in large be applied in an integrated production system.

2 Management information systems

A key issue for companies and organizations to be competitive is their ability to use *information*. Information today is a resource factor, just as humans, raw material, etc. are resources. It is the way information is utilized that determines whether a company or organization is successful or not.

A popular term in this context is *management information systems* (MIS) [Lon89]. This term can be applied to all kinds of information processing systems that support managers at different levels with supportive information.

2.1 Decisions

A critical term often referred to in this context is *decisions*. At many levels in an organization, decisions have to be made. Decisions can involve anything from deciding development of a new car model to deciding to replace a chip in a digital telephone switch. All decisions are based on some form of foundation. This foundation consists of personal skills, knowledge and experience, and various kinds of information. Now, there is simple relationship between the quality of a decision and its foundation. A knowledgeable manager who has all necessary information available can make high-quality decisions. A less knowledgeable manager to whom the needed information is not available cannot make high-quality decisions. The quality of a decision, in turn, is directly related to the overall performance of the organization.

An interesting paradox is that a manager who always makes the right decision is not doing his or hers job properly. The reason for this statement is that if a manager waits until all needed information becomes available, then it is much too late. This has a clearly negative impact in the organization's performance. Thus, in order to contribute positively, decisions must be of a high quality at the same time as they are made at the right time. This means that decision makers base their decisions on incomplete information, and that they have to do the best they can with it anyway.

2.2 Decision support systems

One of the main tasks of MIS is to provide decision makers with the information that they base their decisions on. The better the information MIS can provide, the better the quality of decisions. The task of the MIS is to collect relevant data, structuring, selecting and sorting it, and to present it in an effective way.

Due to the above, sometimes the term *decision support systems* (DSS) is used interchangeably with MIS. According to some sources, however, there is a distinction which should be noted. MIS applications generally works with highly structured data. In particular for high-level decision makers, the necessary information is less structured (or even is totally unstructured). DSS is said to be concerned with less structured information.

Since there is an obvious relationship between the quality of information from an MIS and the overall success of an organization, it is clearly interesting to work on improving MIS. This fact holds both for MIS, working with structured information, and DSS, working with less structured information. In the rest of this document, for the sake of brevity, we will only use the term MIS although the reasoning applies to both MIS and DSS

2.3 Functional vs. integrated MIS

A few years ago, many organizations used *functional* MIS. This means that a number of MIS applications are used independent of each other. Each MIS usually contains some form of database. Due to the independence, the information of the organization must be distributed over a large number of MIS. Moreover, information needs to be duplicated for several MIS, which implies risks for inconsistencies as well as high maintenance costs.

Today, most organizations are turning to an *integrated* view of MIS. This means that the functional MIS elements are connected so that they can share data, cooperate etc. As a result, corporate data is easier to maintain, accessibility is greatly improved and the information “quality” is much improved. The integrated view of MIS is supported by powerful database systems, communications networks, and integrated software.

2.4 Elevating MIS

There are two directions which should be pursued in order to improve MIS. One direction is in *information processing*. Improving information processing leads to higher quality information to decision makers. It may also contribute in shortening the time needed to provide the relevant information, and to reduce possible information overloading. The other direction to be pursued concerns *information presentation*. Using graphical user interfaces, it is now possible to present information in a more effective way. Even though the output of an MIS may be of a very high quality, it needs to be *communicated* to the decision maker in a very efficient way.

3 Relational databases

The key in most MIS applications is the *database*, both as a source of information for processing and as a means to integrate different MIS functions in the organization. Today, relational database systems [Ull82] are in widespread use in most such applications. Relational databases feature great flexibility in data representation and retrieval, while providing a high degree of efficiency.

Before proceeding with the actual connection between SICStus Prolog and MIS, we will take a somewhat deeper look into the fields of relational database technology and deductive database technology. These two sections are more formal than the rest of the document.

3.1 Relational theory

The theoretical foundations of relational databases is *relational theory*. This theory describes how information can be represented, accessed and interpreted.

A *relation* R between a finite set of objects x_1, x_2, \dots, x_n belonging to domains D_1, D_2, \dots, D_n is a subset of the cross product of the domains:

$$R \subseteq D_1 \times D_2 \times \dots \times D_n$$

An *instance* of the relation R is a sequence of objects a_1, a_2, \dots, a_n such that $\langle a_1, a_2, \dots, a_n \rangle \in R$. Such an instance is called a *tuple* and is sometimes written as $r(a_1, a_2, \dots, a_n)$. The elements of a tuple are called *attributes*.

It should be noticed that relations, at least in relational databases, are defined in an *extensional* way, that is, by enumerating the actual objects for which the relation holds. Relations are not defined *intentionally*, e.g., by expressing relations by formal constraints.

There are two fundamental operations for manipulating relations and the data contained within them. These operations always result in a new relation. In combination with these two operations, a third principal operation is often used.

The first operation is *projection*, where a subset of the attributes of a relation are selected. Formally, projection can be described as a function $\pi : R_1 \mapsto R_2$, where the set of attributes A_{R_2} of R_2 is a subset of the set of attributes A_{R_1} of R_1 : $A_{R_2} \subseteq A_{R_1}$.

The second operation is used to combine different tables. This operation is called *join* and can be described as a function $\bowtie : R_1 \times R_2 \mapsto R_3$. The function takes two relations R_1 and R_2 as arguments and returns a relation R_3 which is a subset of the cross product of the other relations $R_3 \subseteq R_1 \times R_2$. There are a number of variants of the join operation, outer join, inner join etc., yielding different results.

The third operation, that is used in combination with particularly projection, is *selection*. The selection operation can informally be described as a function $\sigma : R_1 \mapsto R_2$, where $R_2 \subseteq R_1$. The selection operation is used to “mask out” the tuples of a relation that satisfies certain conditions.

Examples: assume that we have two relations

$$\begin{aligned} R_1 &\equiv \{\langle a, 7 \rangle, \langle e, 2 \rangle, \langle c, 9 \rangle, \langle b, 4 \rangle\} \\ R_2 &\equiv \{\langle 55, g \rangle, \langle 9, y \rangle, \langle 18, u \rangle\} \end{aligned}$$

An example of a project operation:

$$\sigma(\langle a_1, a_2, a_1 \rangle, R_1) = \{\langle a, 7, a \rangle, \langle e, 2, e \rangle, \langle b, 4, b \rangle\}$$

An example of a join operation:

$$\bowtie (R_1, R_2) = \{\langle c, 9, 9, y \rangle\}$$

3.2 Relational database terminology

Although the same foundations are used when talking about relational databases, a somewhat different terminology is used. A relation is often called a *table*. A database may contain any number of relations, which means that it may consist of any number of tables. The attributes of a relation are called the *columns* of a table. The relation instances are called *rows* of a table.

The primitive operations of selection and join are also used in relational database terminology. These primitives provide the means for accessing a database. In a relational database, it is also possible to update, insert and delete tuples (i.e., rows).

3.3 SQL

A number of *query languages* for relational databases have been developed. Today, the most common such language is SQL (Structured Query Language). The language is based on relational theory and has been standardized in different versions.

SQL contains a number of primitive functions, which maps a number of database tables onto a single database table. For example, a selection statement can have the following appearance:

```
SELECT FOO, BAR FROM FROTZ WHERE FOO > 10 AND BAR < 10
```

There is no explicit join operation. Instead, join operations are expressed implicitly in the conditions of a SELECT statement.

4 Deductive databases

In the early 1970s, researchers in logic and logic programming begun looking at predicate logic as a means for managing databases [Gal78, Llo83, Min88, Tär78]. The goal was to build a database language using a formal language like first-order predicate logic. The principal operation on such a language is *deduction*. This, in effect, means that accessing a database would correspond to proving theorems. In turn, this would result in more powerful database applications.

The principal difference between conventional relational databases and deductive databases is that the former only allows extensional definitions. Deductive databases, on the other hand, allows both extensional and intensional definitions. This means that relations can be described by more general expressions— not only by enumerating all instances within a relation.

Example: assume that we want to define a relation *factorial*. In a relational database, we need to define the relation by enumerating all instances, e.g.

$$\text{factorial} \equiv \{\langle 0, 1 \rangle, \langle 1, 1 \rangle, \langle 2, 2 \rangle, \langle 3, 6 \rangle, \langle 4, 24 \rangle, \dots\}$$

In a deductive database, it would be possible to define this relation intensionally, by expressing a general statement about the relation.

$$\forall x \forall y \text{ factorial}(x, y) \leftrightarrow [x = 0 \wedge y = 1] \vee \exists x' \exists y' [x' = x - 1 \wedge \text{factorial}(x', y') \wedge y = y' x]$$

It should be noted that it is possible to make intensional definitions of infinite relations, which of course is not possible using extensional definitions.

Another principal difference between relational databases and deductive databases is the limitation in expression of instances. Relational databases only allow *ground* instances of a relation to be stored. This means that the represented data is in a sense “fixed,” that is, does not contain any variables. In contrast, a deductive database would have the capability to store variables within a relation. Using variables, it is possible to express certain general properties, thus allowing a single specific instance describe a wider class of instances.

In order to access data in a deductive database, a theorem proving program for predicate logic is used. Determining whether a given tuple t is an instance of a relation R corresponds to a formal proof of $t \in R$. This, in turn, can be written as $\vdash R \rightarrow t$. More complicated queries can be expressed as non-atomic predicate logic formulas. For example, assume that we have two relations: *fact* containing two attributes $\langle x, y \rangle$, where y is the factorial of x , and a relation *prime* containing a single attribute $\langle z \rangle$, where z is a prime number. Now, a query for all factorials of prime numbers less than 1000 can be written as:

$$fless(y) \leftarrow \{fact(x, y) \wedge prime(x) \wedge x < 1000\}$$

Although intensive research in deductive databases, there is still much that has to be done before deductive database technology can be used in industrial applications. One of the foremost deductive database projects is Datalog [Ull88] at ECRC in München. Datalog is a language — a subset of first order predicate logic — which has been limited in expressiveness to allow efficient execution. The current limitations is that relation instances must be ground, and that functions are not allowed.

4.1 Relational databases and logic programming

A promising direction for extending the database capabilities is to combine conventional relational database technology with a logic programming system. In practice, this approach means that a logic programming system is integrated with a relational database manager.

In such a hybrid system, extensional definitions of data are held in the relational database, while intensional definitions are held within the logic programming system. Retaining a conventional relational database also means that a number of solutions to technical problems can be retained. This includes, for example, multi-user systems, concurrency control, backup mechanisms and security issues.

The above approach can only be successful if the integration is “tight,” that is, technical issues as data type conversion, memory management, etc. should all be hidden to the user. In terms of a logic programming language as e.g., Prolog, the methods for accessing the database, data types etc. must all be within the language “paradigm.”

5 Databases and SICStus Prolog

In SICStus Prolog, a database system was developed as a library to the Prolog system to allow external storage of terms. This database can be viewed as a deductive database. The data format used are ordinary Prolog terms. A set of built-in predicates are provided to handle the various operations on the databases.

Using a dynamic indexing technique, access is very fast, even for large databases. In contrast to conventional databases, stored data may contain variables, which increases expressiveness.

Another contributing factor is that, unlike Datalog, the SICStus Prolog database handles terms that include functions. In short, all Prolog terms can be stored as elements in the SICStus Prolog database.

5.1 A different view

It should be noted that there are many important differences between relational databases and the SICStus database. In the latter case, a rather different view of databases is used.

In the SICStus database, there is no physical correspondence to tables. Instead, a convention is used to represent tables and attributes. A table is represented as a Prolog *term*. The term consists of a functor and a sequence of arguments. The functor denotes the table name, while the arguments denote the columns. Thus, a table is viewed as a term like, e.g., *persons(Name, Address, Zip, Phone)*.

5.2 Defining a database

Tables need not be defined explicitly. Instead, a table becomes “defined” as soon as an instance of a relation is stored in a database. A “database” is a physical file which holds the data. Several databases may be open simultaneously and data may be spread over several databases.

In order to improve the efficiency, an index model need to be supplied. The index model describes which attributes are used as indices. Using too few indices results in compact databases, but poor performance. Too many indices, on the other hand, results in very large databases, but with good performance. In practice, it is necessary to decide upon the smallest set of indices which provide sufficient performance. In general, those attributes involved in queries to the database should be indexed.

The index model is provided when creating a database. Only one index model can be used for a database. Thus, if two different relations require different index models, then two separate databases (i.e., database files) should be used. Unfortunately, in the current version, the database has several limitations. One of them is that index models may not be too complicated — only a few attributes can be included in an index model.

5.3 Accessing a database

There are two basic operations that can be performed on a database: querying the database for information, and updating the database with new information.

Querying is accomplished by using predefined predicates. There are two basic ways for querying the database. A *template term* can be used to “mask out” the data in the database. In this case, the unification mechanism matches the template term with the instances in the database. The first matching instance is returned as the result. In order to find all instances matching the template term, the Prolog primitives `setof`, `bagof` or `findall` can be used.

The other way a database can be queried is through a direct reference to a relation instance in a database. Each term in the database has a unique reference — *TermRef* — from which the exact physical location of the term can be determined. This reference corresponds to the ROWID attribute used in some relational databases. Of course, the reference must be known prior to querying. The reference can be determined when doing a “pattern matching query” or when the term is inserted into the database.

Terms are inserted into a database using a predefined predicate. As a result of the insertion, a reference to the term is returned. There is no direct support for updating a term. Instead, the term to be updated must first be accessed, deleted and then the modified term can be inserted into the database.

5.4 Current limitations

The current version of the SICStus Prolog database has several limitations which, in particular, have important implications for application development. In this section, we describe some of the limitations, their impact on application development and possible remedies to overcome them.

5.4.1 Wide tables

The SICStus database cannot handle complex index models. This means that “wide” database tables are difficult to handle. For example, if a table contains 20 attributes, then only five of them can be used as indices. In relational databases, any form and number of indexing can be used.

A possible remedy for this problem is to split wide tables vertically. This is, in practice, only possible if the *primary key*¹ consist of a single or, at most, two attributes. Assume, for instance, that *NAME* is a primary key in the table:

```
person(NAME, ADDRESS, ZIP, PHONE)
```

This table can be split into two tables that can be joined over the *NAME* attribute:

```
person1(NAME, ADDRESS, ZIP)
person2(NAME, PHONE)
```

5.4.2 Table browsing

In most MIS applications based on a relational database, it is possible to “browse” database contents in order to find a certain item. Since a query to a relational database always returns a table — in the simplest case a table of one row and one column — it is possible to efficiently retrieve a set of records that can be investigated.

The SICStus Prolog database is very efficient in accessing a single piece of data, that is, a single term. In the current version, querying for a set of terms is made by using a general template term and one of Prolog’s set-of-solutions control predicate. This operation, however, is not very efficient. Due to this inefficiency, the SICStus Prolog database is ruled out as “database” in e.g. MIS applications.

There is no obvious remedy to this deficiency. Hopefully, this problem will be addressed in the continuing support and development of SICStus Prolog.

5.4.3 Joins

The key operation in relational databases is the join between tables. This operation provides means for linking information. In contrast to e.g., hierarchical or network databases, the linkage is “soft,” not controlled by physical references. This dynamic feature of relational databases provide powerful means for structuring and maintaining large and complex collections of data.

Join operations are easily and naturally expressed in SICStus Prolog. Assume, for instance, that we want to join the two tables *person1* and *person2* above. Join is expressed through logical variables. A query would have the form:

```
fetch(person1(NAME, ADDRESS, ZIP)), fetch(person2(NAME, PHONE)).
```

The shared variable constrains the database search to the instances where the *NAME* attribute is equal. This type of join is sometimes called *equijoin* and is by far the most common relational database operation.

Although it can be elegantly expressed in SICStus Prolog, there are severe problems with performance. The operation simply takes too long.

¹A primary key is a set of attributes which is sufficient to “point out” each row in the table.

There is a possible remedy to the problem, based on network database technology. If all necessary join operations are known in advance, then separate *join-tables* can be constructed. Such a join-table contains pairs of term references to two (or more) relations, and an entry for the join element. At run-time, the join-table is accessed in whenever a join operation is to be performed. The resulting set of terms directly point out the terms that the join operation results in.

There is an obvious drawback to this solution. If the tables to be joined contain m and n instances, respectively, then a join-table for the two might contain up to mn instances. In MIS applications, tables often contains many thousand rows, which results in very large join-tables.

5.4.4 Databases

Since the SICStus Prolog database requires “slim” relations and join-tables, the only way that MIS applications can be developed is by using a rather large number of relations. Since only one index model can be used for each database, it is therefore necessary to use a large number of databases. Unfortunately, it is not possible to hold a large number of databases open simultaneously.²

5.4.5 Distributed use

Since the integrated view on MIS is dominant today, it is necessary for several users to simultaneously access data in databases. Most relational database systems support this in an elegant and efficient way. This is not the case for SICStus Prolog. Since databases correspond to physical files on disk, it is possible for several users to access the databases. The problem is that only one user at a time can access a database. In practice, an entire database is *locked* for exclusive use when opened by a user.

5.4.6 Conclusions

The SICStus Prolog database alone cannot meet the demands of MIS applications. The drawbacks and deficiencies are too severe to overcome. Instead, a conventional relational database should be used, in combination with SICStus Prolog. Then, the SICStus Prolog database can be used as a complement to the relational database. Such a combination can contribute to more efficient and advanced MIS applications.

6 Using Prolog to elevate MIS applications

In the foreseeable future, we cannot expect that Prolog becomes a major programming language for MIS applications. The obvious reason for this is that Prolog simply is not suited for being the kernel programming language in MIS applications. It is, however, interesting to note that Prolog — when used as a complement to “ordinary” programming tools — can contribute in a functional elevation of MIS applications. Due to its graphical user interface, SICStus Prolog can also contribute in making more efficient user interfaces.

The aim of MIS applications is to provide the right information to the right person at the right time. Since decision-making using MIS applications is one of the most important factors in a company’s competitiveness, better MIS applications result in better decisions, which in turn results in better business. Using Prolog in MIS applications contributes in three directions:

- The information quality can be elevated using SICStus Prolog and techniques from the field of artificial intelligence.
- The performance can be improved using SICStus Prolog, for example to make a MIS application adaptive.

²The possible number of open databases is mainly dependent on the amount of primary memory available.

- The user interface can be improved using the graphics facilities of SICStus Prolog.

In this section, we describe some interesting techniques in the current context. Using these techniques, implemented in Prolog or any other programming formalism, it is possible to elevate MIS applications.

6.1 Rule-based programming

Rule-based programming is the main programming formalism for present knowledge based systems. A large number of successful knowledge based systems have been produced and such applications are in daily use all over the world.

One reason for the success of knowledge based systems is that the programming technique used is tailored to tighten the “gap” between formalisms that computers can understand and our own way of solving problems. Thus, it has become easier to express human problem-solving knowledge in the forms of computer programs. This, in turn, has made it possible for MIS applications to produce information of a higher “quality.”

The programming formalism used in most knowledge based systems is called *rule-based programming*. Such a program consists of a set of rules. The rules consists of two or three parts: a set of *preconditions*, a *hypothesis*, and a set of *actions*.³ If the preconditions of a rule are all satisfied, then the rule can be *fired*. This means that the hypothesis is given the value “true” and the action statements are executed.

Rules can be executed in two basic ways: by *forward chaining* and by *backward chaining*. In forward chaining, a rule is triggered by change in data. This can be viewed as a form of data driven execution. No specific “execution path” is followed. In backward chaining, a “proof” of a specific hypothesis is attempted. The preconditions of a rule with the given hypothesis are then evaluated. This may result in new hypotheses to “prove” etc. In this way, execution can be controlled — following specific problem-solving “paths.” In many systems, both modes for execution are used together.

There are many specialized tools available for this kind of programming. These tools often combine rule-based programming with object-oriented programming. They are therefore called *hybrid tools*, since they mix two formalisms.

Rule-based programming can also be made using Prolog. Prolog can be directly viewed as a backward chaining rule-based programming system. Forward chaining, however, is more complicated to accomplish. A benefit with Prolog is that “knowledge programming” can be easily combined with “algorithmic programming,” which is less clear in most hybrid tools.

6.2 Explanation-based learning

Computer programs in general are static entities — they do not improve their performance or “skills” over time. An interesting feature of Prolog is that it is rather easy to create adaptive applications. A special technique, called *explanation-based learning*, (EBL) [Kra89] allows a program to improve its performance over time. Moreover, this technique is rather simple, which makes it more feasible for actual implementation. It should be noted, however, that explanation-based learning is very difficult to use with other programming formalisms.

A program execution in Prolog is initiated by giving Prolog a conjunction of goals to prove. During the execution, the Prolog system constructs a *proof tree*. The internal nodes of the tree represent higher-level concepts, defined by lower-level concepts. The latter can be found in the leafs of the tree. The basic idea is to use the execution tree and find a faster way to solve a similar problem next time it occurs.

When an execution has been completed, then low-level concepts are used to form a new Prolog rule. If an identical execution is made, then all internal nodes in the proof tree can be bypassed

³In the case of rules with only two parts, in general the hypothesis part is absent.

by using the special rule instead. The rule can also be generalized by substituting actual pieces of data by variables.

```
p(X) :- q(X),r(X).
q(4).
q(Y) :- t(Y).
r(3).
t(3).
```

Assume that we run the above program by supplying the goal `p(3)`. The execution results in the following tree:

```
      p(3)
     /  \
    q(3)  r(3)
     /
    t(3)
```

Using information from this execution, a new Prolog rule can be defined:

```
p(3) :- t(3),r(3).
```

It is important to point out that all changes to the program should be *additive*, that is, all previous parts of the program are retained unchanged. Explanation-based learning can be used at several levels in a program execution.

The benefit of explanation-based learning is to successively speed up program execution. This is useful for large programs, e.g., a program which computes a sales prognosis. Thus, in MIS terms, EBL contributes by providing the right information in a faster way.

6.3 Partial evaluation

Another benefit of a formal programming language like SICStus Prolog is that it is possible to *precompile* a program before execution, thus making it more efficient. This process is referred to as *partial evaluation*. During the process of partial evaluation, much of the computations that are normally done at run time are instead done at compile time.

Partial evaluation is particularly suitable for declarative high-level programming languages. In such languages, programs can be expressed in a very clear and compact way. The price that has to be paid for this is efficiency. Using partial evaluation, it is possible to retain a clear and compact way of writing programs. These programs are then automatically converted into equivalent but more efficient programs. An example of a partial evaluator is Mixtus [Sal91] which operates on SICStus Prolog programs.

6.4 Case-based reasoning

Case-based reasoning (CBR) [Kol87] is a relatively new technique for construction of knowledge-based systems and is an interesting alternative to rule-based programming systems. Instead of formalizing a problem area as objects and rules, CBR is based on the idea of using a large set of previous solved examples.

There are several important benefits of case-based reasoning systems over rule-based systems. Knowledge acquisition becomes easier since examples are used directly. CBR systems can also be made adaptive over time, widening the scope of the system. Finally, CBR systems have proved to be, at least in some applications, more accurate than rule-based systems.

The main idea is to collect a large database of previously solved problems. When a new problem is encountered, the CBR system tries to find previously solved problems that are as similar to the

new problem as possible. The previous solutions are then used to form a solution to the new problem.

Case-based reasoning requires that problems and the problem domain has certain characteristics. The problem domain must be well-defined and relatively static. Highly dynamic problem domains are, in general, difficult to handle by information systems, and in particular, systems based on CBR. Information must be strictly structured. It is, for example, difficult to base CBR on information in natural languages. Moreover, there must be a substantial amount of information available to the system. The more examples that a CBR system uses, the better the solutions are.

A key concept in case-based reasoning is *similarity*. How do we determine whether one case is more similar to the problem at hand than another? There are several ways to approach this issue. In most CBR systems, a two-step process is used. First, a *primary index* is used to mask out a smaller set of cases from the case base. This index can be said to use a very “rough” similarity measure. The second step is to rank the retrieved cases according to their similarity with the problem at hand. Usually, a weighted sum from the attributes in each case is computed. The highest-ranked case(s) is then used in solution.

A solution can be provided directly, using the solution of a previous case as the solution to the new problem. In more sophisticated systems, a new solution is formed, indirectly based upon the solution of the retrieved case(s).

Case-based reasoning systems are relatively easy to implement in Prolog. SICStus Prolog, in particular, is well suited for this type of technique. The SICStus Prolog database can be used as a complement to a relational database in establishing and maintaining a case base. The pattern matching capabilities are well suited, e.g., for managing primary indices.

Case-based reasoning is applicable in many decision support systems. When making a new decision, it is often fruitful to use information from previously made, similar situations. Thus, again in MIS terms, the “quality” of the information from MIS applications can be improved. CBR can also be used to reduce information overload. In such situations, a system should gather the smallest set of relevant information for a particular task.

6.5 Communicative interfaces

The graphics facilities provided by SICStus Prolog makes it relatively easy to construct *communicative interfaces* to MIS applications. Such interfaces use a mixture of graphical objects and text in order to present information in an efficient way to a user.

Since SICStus Prolog runs on computers with the UNIX operating system, the natural graphical interface needs to be based on X-windows. This windowing system, however, requires lots of low-level programming. The graphics manager in SICStus Prolog supports creation of graphical interfaces at a much higher level. This means that it is very easy to construct interfaces, at least up to some level of complexity.

A few commercial interface construction tools for X-windows are available on the market today. A couple of examples are TeleUse and Open Interface Toolkit. An interesting further development of SICStus Prolog would be to interface SICStus with these interface construction tools.

7 A fault analysis database

The Swedish telecom network today is based on modern switching techniques and digital AXE systems. The technology transition have moved problems from a electromechanical nature to a software-related nature. In order to manage such technology, Swedish Telecom have established a group of highly specialized AXE software experts in Östersund. These experts act as an information center for the field technicians. They also act as a “gateway” between field technicians and construction departements at both Swedish Telecom and Ericsson Telecom AB.

Today, the experts use a support system called AXESS, which is a simple database application. The database contains various information about faults, fault reports, software corrections etc. The application is written using the 4GL tool Nectar and uses a Mimer 3.1 database.

7.1 The information flow

When a field technician encounters a fault in an AXE system, he/she sends a written report to the Östersund group. This report is often accompanied by other information, as e.g., hex code dumps. The report is then sent by ordinary mail.

When the fault report reaches Östersund, a clerk enters the data into the AXESS database. This information is then used by the experts when analyzing the faults.

In the analysis phase, or filtering phase, an expert first tries to find out whether a fault report corresponds to a previously known problem. In this work, the AXESS database is consulted, often by "browsing around" in the database. The aim of the filtering is to be able to provide information about previously handled problems directly. Today, this process takes up to three months due to scaresness of AXE experts, work overload and deficiencies in AXESS.

Fault reports that do not have a correspondence with previously known problems are analyzed. A more detailed fault report is passed on to the construction department in either Swedish Telecom or Ericsson Telecom AB, depending on the faulty subsystem. When the fault has been handled, the construction department passes correction information back to Östersund who, in turn, passes the information to the field technicians.

7.2 The AXESS application

Viewed as a decision support system, the AXESS application does not fulfill the requirements generally made on MIS applications. It lacks some fundamental functionality and is far too rigid. Therefore, it is not possible to incorporate new problem solving techniques to improve the fault filtering task.

7.3 The PAFF prototype

The aim of the PAFF (Prolog-based AXE Fault Filtering) subproject was to investigate whether and how it is possible to improve the decision support system. Another aim of the subproject has been to evaluate SICStus Prolog in a data-intensive MIS application.

The subproject consists of five phases, of which only four could be completed as planned. They are:

1. Problem study.
2. Acquisition of information and data.
3. Application development.
4. Incorporation of case-based reasoning techniques.
5. Evaluation.

7.3.1 Problem study

In the problem study phase, the domain of fault filtering for AXE systems were studied. To accomplish this, a number of visits to the experts in Östersund were undertaken. Continuous discussions led to a good understanding of the problem domain, which has been very useful in the project. The AXESS application and the underlying database system were also studied.

7.3.2 Acquisition of information and data

Although the prototype was not intended to be integrated with the AXESS database, we chose to use the underlying data model together with a database dump — approximately 25 megabytes of data.

The database dump was converted to a format suitable for SICStus Prolog. An attempt was then made to move the data into the SICStus Prolog database subsystem. In the course of this transaction, several severe problems with the SICStus database system were encountered. Unfortunately, these problems made it necessary to completely abandon the SICStus Prolog database in the project. At this moment, the source of the problems has not been determined; it is either the database system in SICStus Prolog, or the UNIX file system or both.

Instead, the internal Prolog database was used. To be able to proceed, the test data had to be reduced to approximately 15% of the original database dump.

7.3.3 Application development

A decision support system for AXE fault filtering experts was developed using SICStus Prolog. The X-windows graphics manager in SICStus Prolog has played a major part in the application. The application resolves many of the problems in the AXESS application. In particular, information browsing — a high-ranked demand from the experts — is much easier in the PAFF application. Moreover, a high degree of flexibility in information retrieval is also provided. An example of a window in PAFF is given in Figure 1.

7.3.4 Incorporation of case-based reasoning techniques

One of the aims of the subproject was to study how case-based reasoning could be utilized in a MIS application. Although the study was made, CBR technology was never incorporated in the PAFF application. The reason for this is that the AXESS database simply does not contain enough data. For example, the important hex code dumps generated by the AXE systems are not included in AXESS. If, however, more detailed information about faults and fault reports is incorporated into the AXESS system, then there is a clear opening for CBR.

The most profitable use of CBR in the fault filtering information is to complement the “normal” means for information retrieval. Using a more flexible access method for data means that information can be obtained on a *relevance*, rather than exact specification basis. For example, given a new fault report, a set of previously processed reports that are similar to the new problem report can be retrieved. Thus, in MIS terms, information quality is improved at the same time as information overload can be reduced.

7.3.5 Evaluation

As usual, an evaluation of the subproject has been made. The evaluation concerns all four “active” phases of the subproject.

7.4 Conclusions

The PAFF applications have given important insights into MIS applications and, in particular, the use of SICStus Prolog in such applications. As a side effect, important insights in the area of fault filtering has been obtained.

A number of problems with SICStus Prolog has been identified, along with some benefits. The derived knowledge is of major importance in the development of MIS applications using SICStus Prolog. It is presented below in the form of hints and guidelines in MIS application development using SICStus Prolog.

Analys av Felrapporter								
<<	>>	Sjk	Beskr.	Nollst(1)	Avsluta			
Felnr : 2156		Reg. dat : 901213						
Handl :		Klar dat :						
TS : V107		Prodnamn : sultr						
Art-nr : CAA 107 4912		Art-rev : r3a						
Prim(r) : SE-B/B2-SWS-0007		Prioritet: b						
Sjkord : RESTART, FC20B, OMS,					Pilotgodk:			
Slogan : IOREI KARD PGA H[NGANDE SLOCI MOT LI3-IND,]TERSTART VID]TERTEST					CNA Frisl:			
Korr :								
CNA :								
Datum	Fel	Slogan						
901213	2156	IOREI KARD PGA H[NGANDE SLOCI MOT LI3-IND,]TERSTART VID]TERTEST						
<<	>>	Sjk	Beskr.	Felleffekt	FR Text	FS Svar	FR hist.	Nollst(1)
FR : SE-B/B2-SWS-0007					Status : svar			
Handl : 901215					Signatur :			
TS : AXE 103 0208/107					Kod :			
Sjkord : RESTART, FC20B, OMS,					Bilagor : b			
Prim(r) :					Prioritet: b			
Fellokalisering								
AP-del :					AP-rev :			
Delsys : bms					Block : sultr			
Art-nr : 2080/CAA 107 4912/237U					Art-rev : r3a			
Felet uppt(ckt vid: DPER								
Datum	Felrapport	Sjkord						
901215	SE-B/B2-SWS-0007	RESTART, FC20B, OMS,						

Figure 1: The main window in the PAFF application.

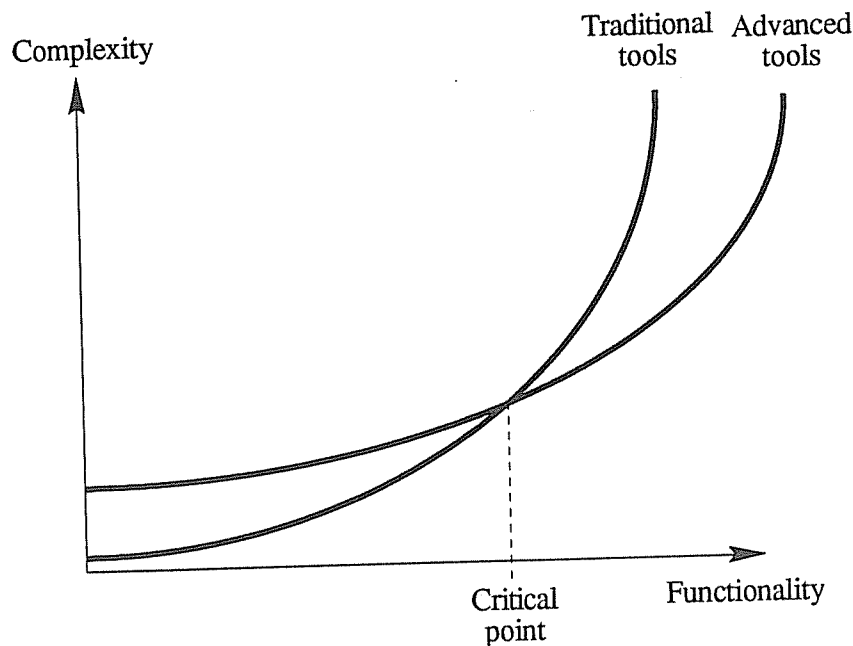


Figure 2: Relationship between traditional and advanced tools.

8 Guidelines in application development

8.1 The role of SICStus prolog in MIS

In order to construct high quality MIS applications, in the sense that they actually do deliver the right information to the right decision maker at the right time, high demands on functionality must be met.

Using traditional application development tools, as e.g., COBOL, C, report generators, forms generators etc, applications with limited functionality can easily be constructed and maintained. As is indicated in Figure 2, there is a "critical point." Introducing more functionality, beyond this critical point, results in applications that are too complex to develop and, more important, are much too complex to maintain.

Advanced application development tools, as e.g., SICStus Prolog are better at handling complex problem solving. Thus, applications with a high functionality can be developed with less complexity in both development and maintenance.

The obvious conclusion from the above is that SICStus Prolog should not be used in MIS application for the part to the left of the critical point. This part is better approached by already available tools and methods. Instead, SICStus Prolog should be used in *combination* with

traditional application development tools. Used as an extension to a “conventional” MIS application, software modules implemented in SICStus Prolog can add the computing power necessary to perform advanced analyses, information retrieval etc.

8.2 SICStus and databases

It is important to have the limitations of the SICStus Prolog database in mind when developing MIS applications. In short, this database should not be used as a database. It is much more beneficial to use conventional relational databases where their functional capacity is sufficient. Instead, the SICStus Prolog database should be used as a *complement* to conventional database management systems. Again, such a combination makes it possible to extend overall system functionality in a less complex way.

8.3 SICStus internal database

The *internal database* of SICStus Prolog, that is, the “program space” is very efficient and can be used as a storage for storing and manipulating substantial amounts of data. This internal database has been used in the PAFF prototype, where approximately 3,5 megabytes of data is stored.

For applications with, say, at least up to 4 megabytes of data to be managed, the internal database may be a realistic choice. The conclusion, thus, is that the internal database can be used for storage and management of large collections of data without problems. The choice, however, is dependent on the type of data and the operations to be performed on the data. Moreover, this also requires that concurrency control is not necessary.

8.4 SICStus and external databases

It is possible to integrate SICStus Prolog with external databases. An interface between SICStus Prolog and Oracle relational databases has been evaluated by Bofors Electronics AB.⁴

SICStus Prolog provides means for integrating Prolog programs with programs written in C. This level of integration, however, is a bit too low in an MIS context. Hopefully, tightly-coupled interfaces for the most common relational databases will be developed in the near future.

8.5 SICStus and rule-based programming

SICStus Prolog is an excellent choice for implementing backward chaining rule-based systems. Due to the simple fact that Prolog *is* a backward chaining rule-based system, there are no difficulties in implementing such applications.

Backward chaining systems are particularly suited to problem solving where *decision trees* can be constructed. Typically, in such a problem, the task is actually to define the problem. This means that initially there is not enough information about the problem. When all information has been gathered, the solution to the problem is obvious. A backward chaining program can be used to collect this information in a well-structured way. Examples of such applications are diagnostic systems, game-playing systems, configuration systems, etc.

SICStus Prolog, however, is not well suited to forward chaining applications. Such applications have a high degree of “asynchronicity.” Large collections of global data control the execution, which does not follow any specific paths. Since one of the features of Prolog is the absence of global data structures, this conclusion is obvious. It is, of course, possible to implement forward chaining in SICStus Prolog. However, this requires a lot of coding that is irrelevant to the problem to be solved. In addition, it is very difficult to achieve efficient forward-chaining systems in Prolog.

⁴Unfortunately, no report of the results have yet been presented.

8.6 SICStus and explanation-based learning

As mentioned earlier, explanation-based learning is a very simple technique in combination with Prolog, but is very hard to utilize for other programming languages and tools.

Explanation-based learning can contribute to an improved performance in situations where problems include heavy data processing, and where different subtasks in the processing are repetitive. To implement EBL in an efficient way with minimum overhead in execution, a somewhat tailored version of the Prolog system is needed. Today, EBL can be implemented as a "layer" on top of the normal Prolog system. This approach does not provide enough efficiency for practical use. A specially tailored version of SICStus Prolog would solve this problem.

8.7 SICStus and case-based reasoning

SICStus Prolog is very well suited for, at least, some forms of case-based reasoning. The pattern matching mechanisms of SICStus Prolog can be used in primary indexing in a very straightforward and simple way. SICStus Prolog can also handle weighted sums etc. in a simple and efficient way. Thus, SICStus Prolog is a good choice for this type of technique.

8.8 SICStus and graphical interfaces

SICStus Prolog provides a high-level interface to X-windows, which makes it very easy to construct graphical interfaces to MIS applications. There are, however, limitations to complexity, which means that the graphics facilities of SICStus Prolog are not appropriate if complex interfaces are to be built.

9 Concluding remarks

Used in its right role, SICStus Prolog can contribute to better MIS applications and, thus, better business. By identifying the benefits and pitfalls of SICStus Prolog in a MIS context, it is easier to see where SICStus "falls in" and where it "falls out." The development of the PAFF application has been of great importance in identifying these benefits and pitfalls.

10 Acknowledgements

We would like to express our gratitude to the ISP "team" at the Swedish Institute for Computer Science: Thomas Sjöland, Stefan Andersson and Kent Boortz. We also thank Ellementel's participant in the project: Hans Nilsson. Johan Sintorn from Mimer Software AB has been a great support in discussions about databases and integration. Finally, we would like to thank all helpful people at Swedish Telecom, including the Östersund group: Örjan Kärnhage and Sture Söderman, and the NFS department: Håkan Lidström and Erik Sparr.

References

- [Col73] Colmerauer, A., Kanoui, H., Pasero, R., Rousell, P., "Un système de communication homme-machine en Français," Research Report, Artificial Intelligence Group, Univ. of Aix-Marseille, Luminy, France, 1973.
- [Gal78] Gallaire, H., Minker, J., *Logic and Databases*, Plenum Publishing Co., New York, 1978.
- [Kol87] Kolodner, J. L., "Extending Problem Solving Capabilities Through Case-Based Inference, *Proceedings of the 4th Annual International Machine Learning Workshop*, 1987.

- [Kow74] Kowalski, R., "Predicate Logic as a Programming Language," *Proceedings of the IFIP Congress*, North-Holland, Stockholm 1974.
- [Kra89] Krawchuk, B. J., Witten, I. H., "Explanation-Based Learning: its Role in Problem Solving," *Journal of Expt. Theor. Artificial Intelligence*, No. 1, pp. 27-49, 1989.
- [Llo83] Lloyd, J. W., "An Introduction to Deductive Database Systems," *The Australian Computer Journal*, 15(2), pp. 52-57, 1983.
- [Lon89] Long, L., *Management Information Systems*, Prentice-Hall, Englewood Cliffs, 1989.
- [Min88] Minker, J., (editor) *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, Los Altos, California, 1988.
- [Sal91] Sahlin, D., *An Automatic Partial Evaluator for Full Prolog*, Ph. D. Dissertation, SICS Dissertation Series 04, ISBN 91-7170-049-8
- [Tär78] Tärnlund, S.-Å., "An Axiomatic Data Base Theory," *Logic and Databases*, (Gallaire, H., Minker, J., eds.), 1978.
- [Ull82] Ullman, J. D., *Principles of Database Systems*, 2nd edition, Computer Science Press, MD, 1982.
- [Ull88] Ullman, J.D., *Principles of Database and Knowledge Base Systems: Volume 1*, Computer Science Press, MD, 1988.