The Columbia-Presbyterian Medical Center Decision-Support System as a Model for Implementing the Arden Syntax

George Hripcsak, James J. Cimino, Stephen B. Johnson, Paul D. Clayton

Center for Medical Informatics, Columbia-Presbyterian Medical Center, New York, NY 10032

Columbia-Presbyterian Medical Center is implementing a decision-support system based on the Arden Syntax for Medical Logic Modules (MLM's). The system uses a compiler-interpreter pair. MLM's are first compiled into pseudo-codes, which are instructions for a virtual machine. The MLM's are then executed using an interpreter that emulates the virtual machine. This design has resulted in increased portability, easier debugging and verification, and more compact compiled MLM's. The time spent interpreting the MLM pseudo-codes has been found to be insignificant compared to database accesses. The compiler, which is written using the tools "lex" and "yacc," optimizes MLM's by minimizing the number of database accesses. The interpreter emulates a stack-oriented machine. A phased implementation of the syntax was used to speed the development of the system.

Introduction

The Arden Syntax is a representation for knowledge bases composed of many independent modules called Medical Logic Modules (MLM's) [1]. Its current focus is on the generation of alerts, management suggestions, management critiques, protocols, and diagnosis scores. The syntax, which is being sponsored by the ASTM international standards committee, is intended to facilitate the sharing of medical knowledge bases. In order to be successful, the syntax must be used at multiple institutions. Several institutions have taken the lead in implementing the syntax: LDS Hospital, Utah; Linkoping University, Linkoping; Erasmus University, Rotterdam; and Columbia-Presbyterian Medical Center (CPMC), New York. At the time of the writing of this paper (March, 1991), an initial version of CPMC's decision-support system, which uses a subset of the Arden Syntax, is running in production on real patient data. A subsequent version of the system, which uses the full Arden Syntax, is running on a test database. This paper reports on the experience we have gained at CPMC while implementing the Arden Syntax.

The overall architecture of the decision-support system has been reported from a software engineering point of view [2]; the focus was on our choice of platforms to achieve the performance goals and the use of software engineering tools. This paper addresses design issues directly related to the Arden Syntax. It focuses on the consequences that the structure of the Arden Syntax has had on the implementation of the system. The syntax has affected us in several ways: the use of p-codes and a virtual machine; the design of the MLM editor, compiler, and interpreter; and the use of a phased implementation. Each of these issues is discussed below in turn.

This work was supported by the International Business Machines Corporation and by a grant from the National Library of Medicine LM04419 (IAIMS).

Use of a Virtual Machine and Pseudo-Code

In the late 1970's one of the factors that led to the rapid spread of Pascal was the use of a compiler-interpreter pair to implement the language [3]. The Pascal source code was first compiled into a list of instructions for a theoretical computer (known as a virtual machine) that did not actually exist. The instructions are known as pseudo-codes (p-codes) because they do not really run on any existing computer.

In order to execute the compiled p-codes, one uses a program called an interpreter on a real computer. The interpreter emulates the virtual machine, interpreting the p-codes into a set of actions on the computer. It is generally small and easy to build.

The main advantage of this method is portability. Moving the system from one computer to a new one requires only rewriting the virtual machine program, which is easier to write than the compiler. Programs compiled on the original computer can be run directly on the new computer because the p-code definition is the same on both.

A second advantage is that the use of intermediate p-codes simplifies debugging and verification of the compiler. It is easier to follow the logic of the virtual machine, which has been designed especially for the task at hand, than to follow the lower level machine language of most computers.

A third advantage is size. Since each p-code can represent a fairly high-level operation, even a large MLM may not require many p-codes. Thus p-code version of a program is often shorter than the corresponding fully compiled version.

The main disadvantage of the use of p-codes is speed of execution, because a program that is compiled into the actual machine language of the computer is faster than one that is compiled into p-codes and interpreted. While this disadvantage is important for languages like Pascal, it is not that important for MLM's, since they spend most of their time performing database queries rather than executing instructions.

When the HELP medical decision-support system [4] was rewritten, the designers chose to use p-codes and a virtual machine [5]. The design was slightly more complex than explained above, since they used a compiler-translator-interpreter triplet. The system easily met their performance goals. (In fact, they found that the p-code version was 30 to 60 times faster than their previous version, which was fully interpreted. There was no fully compiled version with which to compare it.)

We have also adopted the p-code method. There are 96 unique p-codes defined in the interpreter at present. Examples include add, equal, tangent, jump, store to register, and call MLM. We were able to run MLM's compiled by the same compiler on two radically different computers (an IBM 3090 mainframe and an IBM personal computer). Debugging and verifying the compiler was relatively straightforward.

All of our compiled MLM's have been smaller than 500 p-codes (1000 bytes). Based upon the amount of C or PL/I code it would take to implement each of the operators, it is estimated that an MLM compiled directly into machine language would have required up to 10,000 bytes.

The loss of speed associated with p-codes (vs. a fully compiled version) has not been significant in our system. Our current production patient database, which is a hierarchical database that has not been optimized for decision-support queries, requires about 40 milliseconds (elapsed time on an IBM 3090 model 200 mainframe) per query. The average execution time of our longest MLM devoid of database queries is only 0.86 milliseconds (elapsed time). Another way to look at the performance is that our initial goal is to execute 100 MLM's per second [2], which means that 10 milliseconds of the central processing unit's time (CPU time) are available per MLM. The average execution time of our longest MLM devoid of database queries is only 0.35 milliseconds (CPU time), or 3.5% of the total. The length of that MLM is 450 p-codes, but because of the presence of "if-then" statements, the whole MLM is never executed; the effective length of the MLM is 150 p-codes. This results in an average execution time of 2.3 microseconds (CPU time) for single p-codes.

Other design alternatives are available. For example, the Arden Syntax would be easy to implement on an object-oriented system. In this case data types would be classes in the object hierarchy, the operators would be implemented as methods, and the p-codes would be replaced by messages. We did not choose this method because of concerns about performance, and the lack of availability of a single object-oriented system across all of our computer platforms (mainframe, RISC machine, and personal computer).

MLM Editor and Query Syntax

An MLM editor is used by the author to create and maintain MLM's. It contains utilities to search through the knowledge base of MLM's using the Arden Syntax's free text slots, keywords, database references, evoking criteria, and generated alerts. When a new MLM is created, the editor supplies a template MLM with the default slots filled in. The editor also provides a syntax checker to verify the MLM while it is being written, and it contains a help facility. We have experimented with each of these features using prototypes written in Smalltalk, APL, and a screen-building tool for a relational database. Unfortunately, all of them have fallen short mainly due to lack of flexibility in the user interface. Thus, our ultimate environment remains to be chosen.

During the writing of the MLM, the author calls the medical entities dictionary (MED) [6], which contains a hierarchy of medical terms that may be used at the institution. These terms are used in the MLM's database queries. The Arden Syntax explicitly defines the aggregation operator and time constraints for a query, but the rest of the query has been left to be defined at each institution [1]. For example, a query for serum potassium might look like this:

```
K := read last(
      {'potassium concentration measurement'
      where specimen = 'serum'}
      where it occurred within the past 1 week);
```

The portion of the query within curly brackets ("{" and "}") is not defined in the Arden Syntax, but instead is specific to our institution. We have chosen to use a syntax that is similar to the Arden Syntax itself, but which permits terms from the MED (e.g., 'serum'). At an institution that already has a patient database and database management system, the institution-specific portion may use the syntax of the existing system. For example, an institution that uses a hierarchical structure similar to that of the HELP system [4] might phrase the same query like this:

```
K := read last(
      {1 5 23 45 6~Serum potassium}
      where it occurred within the past 1 week);
```

Compiler

The MLM compiler converts the Arden Syntax MLM into a form that can be interpreted in the run-time environment. This requires several steps. In the first step, the native MLM is parsed and converted into a data structure known as a parse tree. The parse tree represents the MLM as a series of nodes in a hierarchy. For example, "var + 3" would be represented by an operator node called "+" which has two children: a variable node called "var" and a number node called "3." This form is easier to manipulate than the native MLM.

Like most institutions that are implementing the Arden Syntax, we are using the tools "lex" (a lexical analyzer generator [7]) and "yacc" (a compiler generator [8]). Even though lex and yacc are not the most sophisticated parsing tools available, they were chosen because they are ubiquitous, resulting in greater portability and easier communication with other groups implementing the syntax.

The next step is to map terms used in the MLM's database queries to actual items in the patient database. In our institution this is accomplished via the metadatabase [9-10], which among other purposes, serves as a data dictionary. Given a high-level database query, the metadatabase returns the codes and procedures necessary to retrieve the data from the patient database at run-time.

Since the database queries take the majority of the available computer time, the goal of the compiler is to minimize the number of queries that actually get executed. Often when an MLM executes, a decision can be made before all of the data are available. For example, if an MLM's alert is never generated when the patient is not hypokalemic, then once it is known that the patient is not hypokalemic, there is no need to read other data. By rearranging the parse tree without changing the actual medical logic, the compiler can improve performance. Further improvements can be gained by accounting for the fact that different queries consume different amounts of time (especially if some of the data are available in buffers); the compiler may be able to perform simpler queries to avoid more complex ones.

The MLM must be converted into a list of p-codes that can be executed by the interpreter. This is done in two steps. First, mnemonic p-codes (similar to assembler instructions) are generated from the parse tree using a C program. Second, they are translated into executable p-codes using a lex program. The two-step process simplifies debugging and validation. It may be done in a single step in the future.

Finally, when a compiled MLM is transferred between computers, a few conversions are done automatically. For example, for some computers, ASCII to EBCDIC conversion must be done for string constants and a format conversion must be done for floating point constants.

Interpreter

Whereas the compiler is evoked once to turn a high-level MLM into p-codes, the interpreter is evoked every time an MLM is executed. Using the MLM p-codes as its instructions, the interpreter retrieves data, processes the data, and generates alerts. The interpreter uses a stack as a temporary storage area. Operations like "add" take their arguments from the stack and return their results to the stack. All of the data types in the Arden Syntax are supported: "null," "Boolean," "number," "time," "duration," "string," and "list." Each item on the stack specifies its own type and value. When an operation is performed, the data types of the arguments are checked first, and then the appropriate action is taken.

Implementing most of the Arden Syntax's operations is straightforward, but operations on the "duration" type deserve special attention. The duration type is really two types: "duration in seconds" and "duration in months." Seconds, minutes, hours, days, and weeks are converted to seconds. Because the number of seconds in a month or year varies with the base date, months and years are treated as a separate type (years are equal to 12 months). The Arden Syntax "time" type is an infinitesimal point that requires a date and time-of-day.

Our interpreter represents a time as the number of seconds that have elapsed since March 1, 1600 (the beginning of this cycle in the Gregorian calendar) expressed as an 8 byte floating point number. This representation was chosen for several reasons. We wanted to be able to represent birthdates from 110 years ago (longer if retrospective data are entered for clinical studies), which does not fit in a signed 4 byte integer. We wanted to use the same format as we use for numbers and durations, which are also 8 byte floating point. The use of floating point means we are not limited to a granularity of one second. March 1, 1600 was chosen simply because it is the most natural starting point for our "time" to date and time-of-day conversion algorithm.

The subtraction of two times always produces a duration in seconds. The addition or subtraction of a time and a duration (seconds or months) produces a time. The subtraction of two times is a simple operation, as is the addition or subtraction of a time and a duration in seconds. The addition or subtraction of a time and a duration in months is more complex. The time is first converted to a date and time-of-day format. Then months are added to the month and year parts of the date. If the result has too many days in a month, then the extra days are truncated. Finally, the date and time-of-day are converted back to the number of seconds since 1600. For example, if we express time in ISO format [11], then when 1 month is added to 1992-01-31T00:00:00, the result is 1992-02-29T00:00:00. Fractional months are added or subtracted by first using the two surrounding integer values and then interpolating. Note that adding a duration in months to a time and then subtracting it will not always result in the original time.

Phased Implementation

Implementing the full Arden Syntax, linking to an institution's patient database, and routing and displaying generated alerts may seem at first to be an overwhelming task. Our approach was first to implement a simple subset of the syntax while the links to the database were being designed. This is not to suggest that institutions should support only a subset of the syntax, but to recommend one reasonable path to implementing it.

One feature that was postponed was the use of the "list" data type [1], which is a variable-length array that can hold zero or more primitive items (number, string, etc.). Although the list data type is considered important — HELP [12] and CARE [13] both support some form of lists — the presence of lists complicates the implementation of the operators.

The syntax supports dynamic data typing [1]. That is, the data type of a given variable in an MLM may not be known until the MLM is executed. The advantage of this is that some patient databases may return different data types for the same query. For example, a query for serum potassium may return either a number or a code like "hemolyzed." In addition, the operators are polymorphic, which means that the same operator can be used in several ways on different data. For example, one can add two numbers to produce a number, and one can add a time and a duration to produce a time. These are really two different procedures that are both known as "add." The result is that operators are complex.

The Arden Syntax's support of dynamic data typing can be postponed. Implementation of the syntax is simplified if one assumes that data types are known at compile-time. This means that a query must somehow specify the type of its result within its institution-specific portion. For example, this would suffice:

```
K := read last(
    {NUMBER
    'potassium concentration measurement'
    where specimen = 'serum'}
    where it occurred within the past 1 week);
```

The type of any variable can then be determined from its first assignment statement. If another assignment statement changes the type, then a compiler error is generated. If a database query tries to return a type other than the declared type, then NULL is returned.

Other features that can be postponed are delayed evocation and MLM nesting. Our first version of the decision-support system lacked each of these features, and we developed a running system very rapidly. It is this system that is currently running in production on real patient data. Since then we have implemented the full Arden Syntax, and we are running it on a test database.

Conclusions

Implementing the Arden Syntax has turned out to be relatively simple and rapid (4 person years for a system running on an IBM 3090 mainframe and IBM PS/2 using OS/2), especially when compared to the tasks of building and populating a patient database, and maintaining a controlled vocabulary. The modular design using a compiler-interpreter pair and the phased implementation have both facilitated the task considerably.

References

[1] Hripcsak G, Clayton PD, Pryor TA, Haug P, Wigertz OB, Van der lei J. The Arden Syntax for Medical Logic Modules. In: Miller RA, ed. Proceedings of the Fourteenth Annual Symposium on Computer Applications in Medical Care. New York: IEEE Computer Society Press, 1990; 200-4.

[2] Hripcsak G, Clayton PD, Cimino JJ, Johnson SB, Friedman C. Medical Decision Support at Columbia-Presbyterian Medical Center. IMIA Working Conference on Software Engineering in Medical Informatics, Amsterdam, The Netherlands, 8-10 October 1990.

[3] Chung KM, Yuen H. A "tiny" Pascal compiler - part 3: p-code to 8080 conversion. BYTE, 1978;3(11):184.

[4] Pryor TA, Gardner RM, Clayton PD, Warner HR. The HELP system. In: Blum BI ed. Information Systems For Patient Care. New York: Springer Verlag, 1984; 109-128.

[5] Ostler MR, Stansfield JD, Pryor TA. A new, efficient version of HELP. In: Ackerman MJ, ed. Proceedings of the Ninth Annual Symposium on Computer Applications in Medical Care. New York: IEEE Computer Society Press, 1985; 296-7.

[6] Cimino JJ, Hripcsak G, Johnson SB, Friedman C, Fink DJ, Clayton PD. UMLS as knowledge base - a rule-based expert system approach to controlled medical vocabulary management. In: Miller RA, ed. Proceedings of the Fourteenth Annual Symposium on Computer Applications in Medical Care. New York: IEEE Computer Society Press, 1990; 175-9.

[7] Lesk ME, Schmidt E. Lex - a lexical analyser generator. Computer Science Technical Report No. 39. Murray Hill: Bell Laboratories, 1975.

[8] Johnson SC. Yacc: yet another compiler compiler. Computer Science Technical Report No. 32. Murray Hill: Bell Laboratories, 1975.

[9] Johnson SB, Cimino JJ, Hripcsak G, Friedman C, Clayton PD. Using metadata to integrate medical knowledge in a clinical information system. In: Miller RA, ed. Proceedings of the Fourteenth Annual Symposium on Computer Applications in Medical Care. New York: IEEE Computer Society Press, 1990; 340-4.

[10] Friedman C, Hripcsak G, Johnson SB, Cimino JJ, Clayton PD. A generalized relational schema for an integrated clinical patient database. In: Miller RA, ed. Proceedings of the Fourteenth Annual Symposium on Computer Applications in Medical Care. New York: IEEE Computer Society Press, 1990; 335-9.

[11] International Organization for Standardization. Data elements and interchange formats - information interchange - representations of dates and times (ISO 8601:1988E). International Organization for Standardization, 1988.

[12] HELP Frame Manual, Version 1.6. Salt Lake City: LDS Hospital, 1989.

[13] McDonald CJ. Action-Oriented Decisions in Ambulatory Medicine. Chicago: Year Book Medical Publishers, 1981.