The Future of Expert System Development Tools?

Jerry T. Ball
Russell F. Moody
ORION International Technologies, Inc.
Presented at the Fifteenth Annual Ideas in Science & Electronics Exposition and Symposium '93

ABSTRACT

This paper describes an evaluation of Expert System Development Tools and the selection of a specific system—ProKappa—for use in two Expert System development projects. Next, we draw on our experience in evaluating and using Expert System Development Tools to make some predictions about the future of these powerful tools.

Expert System Development Tools are powerful programming environments originally intended for use in Expert Systems development. Current trends suggest that the continued success of such tools may depend on their transition to use as advanced software development environments which happen to provide tools for Expert Systems development in addition to tools for the development of more general purpose software. It remains to be seen which Expert System Development Tools will be able to make this transition and which tools will remain viable in the narrower Expert System development niche.

Introduction.

The successful development of an ever increasing number of Expert Systems, or Knowledge-Based Systems (KBS), represents a major commercial application of Artificial Intelligence (AI) research. Expert Systems typically address complex problems requiring considerable expertise to solve (less complex problems are better solved using more traditional procedural programming techniques). The required expertise is usually encoded in a Knowledge Base, often in the form of rules and facts, and a reasoning mechanism, called an Inference Engine, is used to access information from the Knowledge Base. The development of an Expert System is a difficult software engineering project. In fact, it is often the case that the problem the Expert System will solve is not fully understood prior to design and development of the Expert System. To overcome the inability to completely specify the nature of the problem in advance, Expert System development typically relies on the use of a rapid prototyping software development methodology. Additionally, Expert System developers are beginning to rely more and more on the use of powerful software development environments, often called Expert System Development Tools, for Expert Systems development. At a recent conference on Knowledge-Based Systems development, the keynote speaker claimed that Expert System Development Tools provide more than a ten-fold improvement in productivity over development in less sophisticated programming environments.

This paper examines the selection and use of an Expert System Development Tool in two ongoing projects at ORION: the Artificial Intelligence Data Base Interpretation (AIDBI) project, a project aimed at providing Expert Systems, Graphical User Interface (GUI), and high level Text Processing capabilities to the Foreign Aerospace Science and Technology Center (FASTC) at Wright-Patterson Air Force Base in Dayton, Ohio; and, the Knowledge Engineering Implementation (KEI) project, a project aimed at providing Expert Systems capabilities to the Air Force Technical Applications Center (AFTAC) at Patrick Air Force Base in Melbourne, Florida. The paper concludes with a discussion of the future of Expert System Development Tools.

Review of Expert System Development Tools.

The AIDBI project was conceived of, in part, as a follow-on to an earlier Expert System development project, the Assistant for Scientific and Technical Analysis (ASTA). Because ASTA was developed before the advent of powerful Expert System Development Tools, the ASTA system was developed using an interpreted version of Lisp with an inference engine, called MRS, coded on top of Lisp. As a result, ASTA is extremely slow. A basic assumption of the AIDBI project was that while it was desirable to reuse the knowledge encoded in the ASTA knowledge base, the use of Lisp and the MRS inference engine should be scrapped in favor of using an Expert System Development Tool. As a result, a key requirement of the AIDBI project was the evaluation and selection of an Expert System Development Tool to be used in development of a new Expert

System. We decided to evaluate Expert System Development Tools in three stages. The initial stage involved a broad range identification of available software packages and a document based evaluation to determine if the software packages identified had the needed functionality to meet the basic requirements for the AIDBI project. This initial state of evaluation focused on examining objective sources of information and tried to avoid relying too heavily on vendor advertising. During this initial stage, the number of software packages to be considered further was reduced significantly. The second stage in the evaluation and selection process involved an examination of the software packages in action. To the extent practicable, we acquired actual copies of the software packages or full-fledged demo versions for evaluation. Completion of this stage of evaluation led to the selection of a very few software packages which met the needs of the AIDBI project. The final stage of the evaluation and selection process focused on a cost/benefit analysis of those Expert System Development Tools which had been determined to meet the needs of the AIDBI project. While we attempted to be systematic in our evaluation of Expert System Development Tools, many non-systematic factors entered into the evaluation (e.g. were we able to obtain a full-fledged version of the software, was information on products available in the sources we reviewed, was the vendor responsive to our questions, etc.). Further, the specific functional and system requirements of the AIDBI project precluded serious consideration of many Expert System Development Tools (e.g. did the software run on a Unix workstation, did the software have a Motif compliant interface, etc.).

Expert System Development Tools can be roughly categorized into three classes based on the range of capabilities they provide: (a) Expert System Development Tools which provide a basic shell (e.g. inference engine and knowledge base support), but no advanced development environment (e.g. source code linked debugger, graphical developer's interface, etc.) and little in the way of GUI development or graphical display capabilities, (b) tools which provide an advanced development environment in addition to the basic shell, but little in the way of GUI development or graphical display capabilities, and (c) tools which provide an advanced development environment and GUI development and graphical display capabilities in addition to the basic Expert System shell.

Because we were specifically looking for an advanced development environment to facilitate development, we eliminated those Expert System Development Tools which do not provide such capabilities from further consideration. However, we did not immediately eliminate those tools which do not provide GUI development or graphical display capabilities, since it is quite possible to combine an Expert System Development Tool with a separate program providing these capabilities. The success or failure of tools in this second category hinges on the degree to which the end user and developer interfaces can be kept distinct (since they will be provided by different software) and on the ease with which the Expert System can be interfaced to GUI and graphics display software. Expert System Development Tools in the third category provide an integrated development environment for the development and maintenance of Expert Systems and end user interfaces. While these tools provide the functionality

needed by the AIDBI program, several of them also contain additional features which are not specifically required and which add considerably to the cost (e.g. G2 provides support for real-time programming and operation over a distributed network). For such systems, determining the potential benefit vs. cost of these additional features becomes important. Figure 1 diagrams several potential configurations of software which provide the essential capabilities of Expert System Development Tools in the third category.

Review of ProKappa.

As a result of our three stage analysis, it was determined that ProKappa (an IntelliCorp product) was the Expert System Development Tool which came closest to providing the functionality needed for the development of the AIDBI program, without providing extraneous functionality which was not needed. ProKappa is an integrated environment for the development of complex software systems. It comes bundled with a rule-based programming language (ProTalk) and a procedural programming language (C), each containing a powerful debugging tool (the ProTalk Workbench and Saber C, respectively), an inference engine which uses the ProTalk language, a text editor (EMACS or VI), two separate GUI development tools (Interface Workbench and Active Images), powerful object-oriented programming capabilities, and a database access system. ProKappa is a descendant of KEE, one of the first commercially available and most successful Expert System shells. Unlike KEE which was programmed in Lisp and originally ran on special purpose hardware (Lisp machines), ProKappa is programmed in C and runs on Unix workstations.

The selection of ProKappa for use in the AIDBI project was based on the assumption that we could adapt the knowledge base of the ASTA system for use in the AIDBI program. However, ASTA was designed as a standalone system which performed analysis while the analyst sat and watched, occasionally providing input. Analyst's were not enthusiastic about using ASTA and expressed a preference for a system in which they controlled the analysis process, rather than one in which a program directed the process. Essentially, they wanted a system which would assist them by providing information when needed, and not a program which prompted them for information when one of its rules required it. This shift in emphasis focused attention on timely and context dependent provision of information (in a user friendly interface) rather than rule processing, and on the GUI development capabilities of ProKappa rather than on its rule processing capabilities. Unfortunately, although ProKappa provides a powerful tool for the development of GUI's (the Interface Workbench), that tool does not provide a full implementation of the Motif widget set. Once the focus of the AIDBI program shifted from rule processing to GUI development, the shortcomings of ProKappa's GUI development system became apparent. The main shortcoming is the limitation of the GUI development system to an implementation of Motif dialog boxes. This means that a Motif main window cannot be created using the GUI development system, and it complicates the development of a truly Motif compliant interface (a requirement of the AIDBI project). In fact, if one were

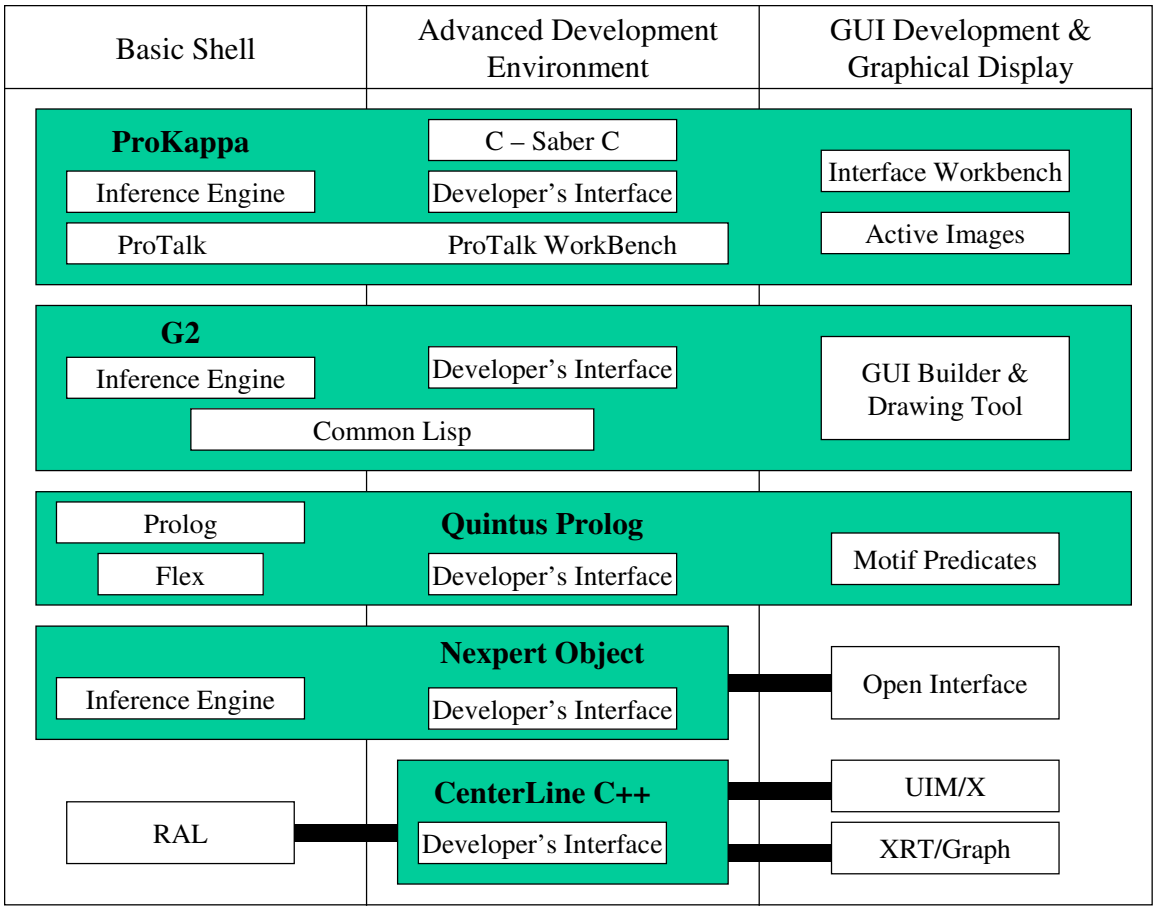

Figure 1: Software Configurations Providing Catetory 3 Capabilities

primarily interested in GUI development, ProKappa would not be a good tool to use since it has many additional features which are not needed and since its GUI development capabilities are limited relative to more powerful GUI development systems which are available (e.g. UIM/X). This holds true despite the fact that IntelliCorp, the developer of ProKappa, is planning a June 1993 release of ProKappa which contains significantly enhanced GUI development capabilities—to include a nearly complete implementation of Motif.

At the same time that the use of ProKappa on the AIDBI project was becoming less appropriate, the Knowledge Engineering Implementation (KEI) project entered its second phase. During Phase I, a feasibility demonstration of a prototype Expert System was developed on a PC using the Nexpert Object Expert System Development Tool. Phase II is focused on the development of a functional prototype, and Phase III will result in development of a fully functional system. The Phase III system must run in a Unix workstation environment. While Nexpert Object comes with a Unix workstation version, we determined that ProKappa provides considerably more functionality than the workstation version of Nexpert Object (which has the same functionality as the PC version) at only marginally greater cost. In fact, Neuron Data, the developer of Nexpert Object, also markets a GUI development product called Open Interface which is intended to be used in the development of GUI's for Nexpert Object. Unfortunately, the combined cost of Nexpert Object and Open Interface is quite prohibitive. Based on the required Phase III implementation on a Unix workstation, we decided to transition to ProKappa for development of the functional prototype in Phase II.

We used ProKappa to develop an initial prototype. The prototype system is required to read in a large amount of data and then analyze that data. Running in the development environment on a Sun SparcStation 1+ with 16 mbytes, the system was taking an inordinate amount of time (as much as 75 minutes) to analyze a single data set. In fact, the system ran so slow that it had a significant and negative impact on development. All 16 mbytes of internal RAM appeared to be used up just to load the ProKappa development environment (causing the system to start swapping to and from the hard disk). To determine how much of the slowdown was attributable to the particular system we were using, we tested the program out on a SparcStation II with 48 mbytes of RAM. The improvement was significant. What had taken 75 minutes on the 1+ with 16 mbytes, ran in 15 minutes on a SparcStation II with 48 mbytes. Clearly, ProKappa requires significant computing resources. When we presented this information to an IntelliCorp representative, we learned that we were probably running our program in the default interpreted C mode! Despite the fact that one of the authors of this article attended an intensive ProKappa training course, we were unaware that ProKappa's default mode was interpreted. In retrospect, it is understandable that the source code linked C debugger requires an interpreted mode, but it is certainly more usual to assume that C code would be compiled. We have since compiled the prototype system and tested it out on three data sets each of which was read in and analyzed in under a minute on a relatively slow Sun IPC (as opposed to 75 minutes).

At least for the time being, the issue of speed has been resolved, although it could resurface as a problem for the fully functional system.

The Future of Expert System Develoment Tools.

Expert System Development Tools are extremely powerful and complex pieces of software. Their use offers great promise for significant productivity improvement, but will such improvement actually be realized? Perhaps the greatest risk to productivity improvement is the risk that these complex pieces of software will contain fatal flaws which ultimately undermine their usefulness.

There are a number of dimensions along which a piece of software can be fatally flawed: speed, stability, cost, limited usability, etc. With reference to speed, the failure of Lisp and Prolog to attract a larger audience outside AI circles is due in large measure to their perceived slowness relative to more procedural languages like C and Fortran. This was certainly true of the older interpreted versions of Lisp and Prolog (witness the speed problems of ASTA mentioned earlier), and it is probably still true (although much less so) of the newer compilable versions of Lisp and Prolog, since these languages make extensive use of recursion and function invocation—two operations which are computationally expensive, and, therefore, slow. In fact, many of the Expert System Development Tools which were originally coded in Lisp have been recently recoded in C to improve their speed (in addition to making them more portable). Nonetheless, Expert Systems Developments Tools are complex programs which introduce a significant amount of overhead relative to coding directly in an underlying programming language like C and speed remains a potential concern.

With reference to stability, the more complex a piece of software is, the less likely it is to be stable. One of the reasons C has become such an important programming language is because it is small and stable (besides being closer to the machine's language, and, therefore, fast). The demise of languages like PL/1 and Algol 68 is largely attributable to their complexity. ADA is having a difficult time becoming an established language despite extensive government support, due primarily to its complexity. One of the authors of this paper suffered through the experience of working with an unstable version of Prolog running on a Lisp machine. Every so often (too often!) the system would just lock up, destroying any changes make since the last save and requiring a reboot that took 10 minutes. Eventually the author gave up on using Prolog on the Lisp machine and moved to a more stable version of Prolog running on a Sun computer. This was no doubt a small event in the history of Lisp machines, but it did portend the problems that were to come.

The fatal flaw of trying to market a piece of software which is overpriced relative to the competition requires little discussion. It should be noted that software development is proceeding at such a rapid rate that any expensive piece of software (including most Expert System Development Tools) is at risk of being undercut by significantly lower priced competitors. Such competitors often bootstrap off or avoid the research and development which was required for the original development. Of course, from the perspective of the software user, price and not loyalty reigns supreme.

Finally, software which is developed for some specific purpose, often fails to be competitive with more general purpose software. This is more true of system level software than it is of specific application software, but Expert System Development Tools are essentially in competition with more general purpose software development environments. For example, CenterLine's software development environment is a powerful environment for the development of C programs (an earlier version of CenterLine's product, called Saber C, is used in the ProKappa system). CenterLine has recently introduced a C++ programming environment which essentially adds object-oriented programming capabilities to the C environment. If there were sufficient demand, CenterLine could also add a rule-based programming module to its environment. Actually, RAL (Rule Application Language) provides just such a C-based rule programming capability, although it has not yet been integrated into an advanced development environment. A RAL + CenterLine C++ combination would provide a general purpose software development environment, along with rule-based and object-oriented programming capabilities, bringing it into direct competition with Expert System Development Tools. Similarly, Quintus markets a powerful software development environment for the development of Prolog programs. Prolog is inherently a rule-based (actually logic-based) programming language and as such competes with many Expert System Development Tools. Quintus has recently added support for X/Motif based GUI development and object-oriented programming (Flex). Quintus also provides hooks to the C programming language (Quintus Prolog is written in C) for the coding of those procedures for which Prolog is inappropriate. In sum, powerful software development environments based on general purpose programming languages (e.g. C, Prolog, Lisp) with added modules to support rule-based and object-oriented programming provide serious competition for the typically less general purpose Expert System Development Tools.

The current trends suggest that Expert System Development Tools will have to become more and more general purpose in order to compete with other software development environments. A contributing factor is the trend towards the embedding of Expert Systems into larger programs, rather than treating the Expert System as a standalone program. In essence, Expert Systems technology is moving into the mainstream of computer software development and is increasingly being viewed as a collection of techniques (e.g. rule-based programming, object-oriented programming, inference engine, knowledge base, etc.) for use in the development of software to solve prolems, rather than as a collection of techniques for solving a specific kind of problem. It will be interesting to see which Expert System Development Tools are successfully transitioned to more general use as advanced software development environments.

Of course, increasing numbers of Expert Systems are being developed (many of which are being developed as standalone programs), and the market for specialized Expert System Development Tools will continue to exist (and perhaps even grow). However, there are currently too many tools available and it is expected that many of these tools will cease to be marketed over the next few years. This reality introduces an added risk to

selection of an Expert Systems Development Tool, namely, the possibility that the tool will not continue to be supported. In fact, several of the tools which were evaluated for use in the AIDBI project are no longer being marketed or supported.

The Future of ProKappa.

How does ProKappa fit in to this rapidly evolving software development environment? ProKappa is no longer marketed as an Expert System Development Tool. Rather, it is marketed as an advanced software development environment with happens to provide rule-based and object-oriented programming capabilities. Perhaps ProKappa's greatest strength is the integrated environment it provides. It is this integrated environment which offers the greatest prospect for improved productivity. In this integrated environment, the software developer is able to choose the right tool for the programming task at hand, without having to worry about integrating that code with code developed using other tools in the integrated environment. In a non-integrated environment, the software developer would spend considerable time and effort interfacing software developed using different (and non-integrated) tools (e.g. GUI development tools, rule-based programming tools, object-oriented programming tools, editors, debuggers, etc.). However, in constructing an integrated environment, some sacrifices were required. Several of the components of the integrated environment provided by ProKappa do not provide all the features of the software products which are available separately. For example, the Interface Workbench (a GUI development component of ProKappa) does not provide a full implementation of the Motif widget set. If one were primarily interested in GUI development, other tools provide greater functionality at less cost.

Despite the shortcomings of the individual components of ProKappa, the integrated environment it provides will lead to significant improvement in productivity, so long as ProKappa does not fall victim to the fatal flaws discussed earlier. In terms of speed, ProKappa programs are for the most part (except for the object base) compilable into C code which is then compilable into machine code. The additional compilation stage introduces a certain amount of overhead (in the form of additional function calls), but the resulting machine code is still likely to be relatively efficient. With regard to stability, the development environment of the current version of ProKappa tends to become unstable after a certain period of time and it is necessary to exit and restart ProKappa occasionally (note that this instability does not cause the system to lock up, but changes due fail to be reflected). Interestingly, this stability problem may be due to a memory leak in the underlying X Window System, and not in ProKappa itself. With respect to cost, ProKappa bundles a lot of powerful software (some of which is admittedly available as free shareware, e.g. EMACS, GCC) at quite modest cost. While there are cheaper Expert System Development Tools currently available, ProKappa is currently very cost/benefit competitive. However, it should be noted that a separate runtime license is required for each seat of a ProKappa developed application (in addition to the development license). As more and more seats are added, ProKappa developed software becomes considerably more expensive than software developed using tools (e.g. C) which do not require a runtime license. Finally, in terms of the

limitations of special purpose software, ProKappa is marketed as an advanced software development environment and not as an Expert System Development Tool. However, given the limitations of some of the software components relative to separately available software products, ProKappa does suffer some limitations as a general purpose tool. If IntelliCorp can reduce the limitations of these software components while maintaining an integrated environment, ProKappa will truly become a powerful advanced software development environment. This may well be a big "if" since it will be difficult for IntelliCorp to keep the capability of its individual components up to the level of the products marketed separately by individual vendors. One way to accomplish this would be for IntelliCorp to become essentially a software integrator, rather than a software developer. To some extent, IntelliCorp has already adopted this posture (witness the use of the Saber C debugging environment, GCC, EMACS, Active Images (a subset of DataViews), and Motif in ProKappa). However, IntelliCorp has not been totally successful in keeping these individual components up to date (in fact, doing so may be prohibitively expensive). Thus, it remains to be seen if ProKappa can remain competitive in the rapidly evolving advanced software development market.