An Evaluation Of Techniques And Tools For Integrating Knowledge-based And Conventional-computing Systems

By

Andrzej Brzezinski

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

University of Manitoba

Winnipeg, Manitoba

ISBN 0-315-85975-X

Name ___

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Computer Science — SUBJECT TERM

0 9 8 4 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523
Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575
Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984
Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

ABSTRACT

Even though Artificial Intelligence (AI) technology became commercially available in 1980s, it was viewed from the wrong perspective throughout most of the decade. A large portion of AI research during this period viewed AI systems as the central part of computing environments. This *AI-centric* perspective led to many limitations in commercially available tools. Recently, AI researchers and developers have begun to view their techniques as extensions to conventional computing environments. This view dictates a need for integration between AI tools and conventional tools. This thesis examines various techniques and tools that have been developed to integrate knowledge-based and conventional computing environments. The topics to be examined include: expert systems and databases, calling external programs from AI tools, embedding AI tools in conventional systems, and techniques for using client-server architectures to support knowledge-based systems. Examples using various commercial tools will be used to illustrate these techniques.

AN EVALUATION OF TECHNIQUES AND TOOLS FOR INTEGRATING KNOWLEDGE-BASED AND CONVENTIONAL-COMPUTING SYSTEMS

BY

ANDRZEJ BRZEZINSKI

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

© 1993

1. Introduction to Expert Systems

1.1. History

An expert system is a special-purpose computer program capable of solving difficult problems in a narrow domain and operating in a manner similar to a human expert. Expert system technology evolved as a result of many years of attempts by Artificial Intelligence (AI) researchers to create real-world AI applications [*Turban, E.,* 1993]. For many years AI research concentrated on general-purpose problem solving algorithms which worked only for very limited, simple problems [*Jackson, P.,* 1986]. Increasing the number of problem states (problem search space) resulted in a combinatorial explosion of possible paths to a solution and general-purpose, exhaustive search algorithms became inefficient and useless. Heuristic search techniques that can be applied to simple problems are elusive in real world problems [*Charniack E., and D. McDermott,* 1985]. The solution to this problem involves using knowledge about a specific problem to guide the search. Knowledge about a specific problem allows excluding a large number of paths which, according to what is known about the problem, offer a very small or no chance of finding a solution. Further research in this direction resulted in development of knowledge representation schemes and reasoning algorithms utilizing that knowledge. As a result, knowledge-based or expert system technology was born [*Van Horn,* 1986].

The first commercial expert systems were developed in the early 1980's and they were the first successful real-world AI applications [*Turban, E., and Watkins, P.,* 1988]. The first expert systems were very limited in their capabilities. They could solve problems within a very well defined area. Knowledge representation techniques and inference mechanisms used in the systems were typically very simple. Often they had to be run on Lisp-machines or other expensive hardware. Usually they were developed in Lisp, Prolog, or other AI-specific programming languages. Dedicated hardware together with specialized programming languages that lacked any interface to the external computing world, resulted in stand-alone systems. In fact, these early systems required little communication with other hardware or software. Throughout the 1980's the number of commercial expert system applications grew substantially [*Schorr, Herbert, and Alain Rappaport,* 1989]. They were becoming more sophisticated in terms of both their capabilities and knowledge representation schemes. In the middle of the eighties, the first expert system development tools capable of running on general-purpose, inexpensive hardware appeared in the market place [*Harmon, P., R. Maus, and W. Morrissey,* 1988]. Expert system technology was becoming cheaper and more affordable. It also resulted in new applications that were economically justifiable on inexpensive hardware [*Quinlan, J. R.,* 1988].

When expert system technology became available on most hardware platforms it soon became clear that this new technology should be able to coexist and cooperate with other components of conventional computing environments. One of the first conventional technologies incorporated into expert system technology was database technology [*Jarke, M. and Y. Vassiliou,* 1984]. The advantages were enormous, expert system applications could access large amounts of data as facts describing characteristics of the problems being solved. They could also store the results of problem solving sessions in a database which could then be accessed and processed further by conventional applications. In the late eighties most expert system tools offered some kind of database interface. They also offered an interface to external functions that could be developed in other conventional programming languages. This further increased the flexibility of expert system applications by giving them access to a rich array of numerical functions found in most general-purpose programming languages like the C language.

Even though expert system technology in the 1980's made unprecedented progress and spread to virtually all computing environments, it still remained fairly isolated. First attempts were made towards closer integration with other elements of computing environments (as described above) but only recently, in the early nineties, have some expert system tools started to evolve in the direction of full integration (e.g. client-server architectures, embedded expert systems) [*Quinlan, J. R.,* 1988].

1.2. Classification of Expert Systems

Expert systems can be classified by the general problem area addressed by the expert system [*Turban, E.,* 1993]. Some expert systems belong to more than one category. The following is a brief description of each category:

Interpretation systems infer situation descriptions from observations. This category includes surveillance, speech understanding, image analysis, signal interpretation, and many kinds of intelligence analysis.

Diagnostic systems include medical, electronic, mechanical, and software diagnosis. Diagnostic systems typically relate observed behavioral irregularities to underlying causes.

Prediction systems include weather forecasting, demographic predictions, economic forecasting, traffic predictions, crop estimates, and military, marketing, or financial forecasting.

Design systems develop configurations of objects that satisfy the constraints of the design problem. Such problems include circuit layout, building design, and plant layout. Design systems construct descriptions of objects in various relationships with one another and verify that these configurations conform to stated constraints.

Planning systems specialize in problems of planning like automatic programming. They also deal with short- and long-term planning in areas such as project management, routing, communications, product development, military applications, and financial planning.

Monitoring systems compare observations of system behavior to expected behavior. An example is monitoring instrument readings in a nuclear reactor to detect potentially dangerous and accident conditions.

Debugging systems rely on planning, design, and prediction capabilities to create specifications or recommendations for correcting a diagnosed problem.

Repair systems develop and execute plans to administer a remedy for some diagnosed problems. Such systems incorporate debugging, planning, and execution capabilities.

Instruction systems incorporate diagnosis and debugging subsystems that specifically address the student as the focus of interest. Typically, these systems begin by constructing a hypothetical description of the student's knowledge that interprets his or her behavior. They then diagnose weaknesses in the student's knowledge and identify appropriate remedies to overcome the deficiencies. Finally, they plan a tutorial interaction intended to deliver remedial knowledge to the student.

Control systems adaptively govern the overall behavior of a system. To accomplish this, the control system must repeatedly interpret the current situation, predict the future, diagnose the causes of anticipated problems, formulate a remedial plan, and monitor its execution to ensure success.

1.3. Benefits and Limitations of Expert System Technology

Expert System technology can provide some or all of the following benefits [*Turban, E.*, 1993]:

Increased Output and Productivity: Expert Systems can work faster than human experts. For example, XCON has enabled DEC to increase fourfold the throughput of VAX configuration orders [*Waterman,* 1986].

Increased Quality: Expert Systems can increase quality by providing consistent advice and reducing the error rate. For example, XCON reduced the error rate of configuring computer orders from 35% to 2% [*Waterman,* 1986].

Reduced Downtime: By using Expert Systems for diagnosing mulfunctions and prescribing repairs it is possible to reduce downtime of a mulfunctioned system or machinery, thus saving significant amounts of money.

Capture of Scarce Expertise: Expert Systems can provide expert advice in situations where there is not enough experts for a task or where an expert is retiring or leaving a job.

Flexibility: Expert Systems can offer flexibility in both services and in manufacturing industries. For example, XCON helped DEC to better fit the variety of customer requests, which was becoming increasingly difficult.

Easier Equipment Operation: Expert Systems can train people to operate complex systems. For example, STEAMER is an Expert System that trains inexperienced workers to operate ship engines [*Waterman,* 1986].

Elimination of the Need for Expensive Equipment: Expert Systems can perform monitoring and control tasks using less expensive equipment because of their ability to investigate more thoroughly and quickly the information provided by instruments. DENDRAL is an example of such an Expert System [*Waterman,* 1986].

Operation in Hazardous Environment: Expert Systems can replace humans operating in hazardous environments, e.g. military tasks during a war.

Accessibility to Knowledge: Expert Systems can make the expert knowledge widely accessible while freeing precious time of human experts for solving difficult problems rather than providing expertise to others.

Reliability: Expert Systems are more reliable than human experts. They do not become bored, tired, or sick.

Increased Capabilities of Other Computerized Systems: Integration of Expert Systems with other computer systems, e.g. databases, makes the other systems more effective: they can work faster, be easier to use, and produce higher quality results.

Integration of Several Experts' Opinions: In some applications, Expert Systems can integrate knowledge of several experts and thus may increase the quality of advice.

Ability to Work with Incomplete or Uncertain Knowledge: In contrast to conventional systems, Expert Systems can, like human experts, work with incomplete information. The user can respond with a "don't know" or "not sure" answer to one or more of the system's questions during a consultation, and the Expert System will still be able to produce an answer, although it may not be a certain one.

Ability to Solve Complex Problems: Some Expert Systems are already capable of solving problems where knowledge required exceeds the scope of any one human expert. In the future, it may be possible to create Expert Systems able to solve problems whose complexity exceeds human ability.

Knowledge Transfer to Remote Locations: One of the greatest potential benefits of using Expert Systems is ease of its transfer across international boundaries.

Expert System methodologies available today are not always effective and impose severe limitations on some applications. Some of the problems with Expert System technology are listed below [*Turban, E.*, 1993]:

- Knowledge is not always available.
- Expertise is hard to extract from humans.
- The approach of each expert to situation assessment may be different, yet correct.
- It is hard, even for a highly skilled expert, to abstract his or her expertise, especially under time pressure.
- Users of Expert Systems have natural cognitive limits.
- Expert Systems work well only in a narrow domain.

- Most human experts have no independent means of verifying their conclusions.
- Help is often required from knowledge engineers who are rare and expensive.
- The end users frequently have no trust in Expert Systems' conclusions.

1.4. Development of an Expert System Application

Development of expert system applications is quite different than development of conventional computing applications. Because of the complex and often unclear nature of problems that an expert system is to solve, it is impossible to create a complete set of design specifications and then implement the system according to them. The only viable approach is incremental development [*Turban, E.*, 1993].

Before development of an expert system application can even start, it is important to do a feasibility study that should answer the following questions:

- What problems should the system be able to solve ?
- What resources are required and what resources are available (most important being an easy access to the source of expertise e.g. a human expert) ?
- What is the likelihood of a failure of the project ?
- What is the estimated development time ?
- Is there an expert system package available that fits the application or should the application be developed from the ground up ?

Only after the above questions have been resolved should development begin. The next step is identification of an initial scope of a prototype. It should not be too wide since developers could easily get lost in the amount of knowledge needed to solve a wide range of problems. Also, knowledge would not be deep enough. On the other hand, an initial scope should not be too narrow because developers would run into problems in the future trying to scale up the system (scaling problem) [*Hayes-Roth, F., et al.*, 1983].

After an initial scope of a prototype has been chosen developers can start extracting knowledge from a source of expertise (which is usually a bottleneck in expert system development). Based on the extracted knowledge, developers can create a conceptual model of a system and map it into a knowledge representation and reasoning scheme. Mapping of the conceptual model into the actual knowledge representation can depend to a large degree on an expert system tool selected for the application. At this point it may turn out that the tool is not appropriate for the application and it should be discarded. It is often very tempting for developers to try to force a conceptual model into the tool which usually results in serious problems later in the development cycle when the scope of the prototype is expanded. Sometimes, especially for large and complex applications the best solution is to build the system from the ground up including the knowledge representation and reasoning scheme. Fortunately expert system tools available in the market place today are becoming more and more flexible and powerful, satisfying the needs of even very complex applications [*Turban, E.*, 1993].

After mapping the conceptual model into the knowledge representation and inference mechanisms developers can start developing the first prototype. The prototype should be thoroughly tested by both developers and the expert to make sure that it can handle all cases within the initial scope of the application.

When the first prototype is approved by the expert the scope of the application can be gradually expanded and the cycle will be repeated. By incrementally increasing the scope of the application developers should not get lost in the excessive amounts of knowledge and can better control the development process. Many of the today's expert system tools are very well suited to incremental development. One of such tools is Level5 Object that offers excellent support for incremental development. It provides developers with a range of facilities such as user interface tools, knowledge management tools, database interfaces, and a very user-friendly development environment [*Level5 Object Reference Guide,*1990].

1.5. Expert System Shells

Expert systems are composed of six basic elements: knowledge acquisition subsystems, inference engine, explanation facility, user interface subsystem, knowledge base management subsystem, and knowledge base [*Turban, E.,* 1993]. The first five components constitute an expert system shell. An expert system shell is in general application independent, so, once constructed it can be reused in many applications. On the other hand, the knowledge base determines what problems an expert system will be able to solve.

By using the shell approach, expert systems can be developed much faster. Furthermore, the programming skills required are much lower. An expert system shell, can be very useful in developing an expert system, providing it is well chosen. There are many different types of expert system shells and each of them has its strengths and limitations. For example, some shells support only rule-based knowledge representation, and in this category some shells provide only backward chaining or only forward chaining inferencing mechanisms. Other shells support only frame-based (object oriented) knowledge representations. Still other, more flexible, shells support both rule-based and frame-based knowledge representations. An application may require a hybrid, both rule-based and frame-based, knowledge representation, so choosing a shell that supports only one of them may create serious problems during development. However, by selecting an expert system shell carefully, the development of an application can be much easier and faster. Many expert system shells provide excellent development tools; for example a debugger, knowledge base editors, user interface tools, and a trace facility that makes the development process even easier.

Apart from these general-purpose expert system shells, there are also domain-specific shells designed for a particular type of application, for example a shell for diagnostic systems [*Turban, E.,*1993]. Domain-specific tools can greatly reduce the risk of choosing the wrong tool for an application. Presently, there are only a few domain-specific expert system shells available and they cost much more than general-purpose ones.

1.6. Summary

Even though AI technology appeared on the commercial market around 1980 it was viewed from the wrong perspective throughout the whole decade [*Earl D. Sacerdoti*, 1989]. AI commercial tools, including expert system tools, of the eighties suffered from the same "disease" as non-commercial AI systems being developed in AI labs. From the very beginning AI researchers viewed AI systems as the central part of computing environments. This "AI-centric" perspective was present in the commercial tools of the eighties and was the primary reason for the limited applicability of those systems. AI researchers failed to realize that AI systems should be viewed as an extension of conventional computing environments and therefore should integrate well with them. The majority of tasks in a real-world computing system can be solved using conventional programming technology. This should be the core of any such system. AI technology can extend the capabilities of a conventional system in some areas, but only if it can be integrated with the conventional environment, take advantage of some conventional functions (e.g. database technology), and satisfy all the rigorous requirements of a conventional environment. In other words an AI system should behave like a conventional program and be able to communicate with conventional elements of the system. Stand-alone AI systems, on the other hand, are very limited in their capabilities and do not satisfy the needs of today's complex systems [*Earl D. Sacerdoti*, 1989].

The purpose of this thesis is to examine techniques and commercial tools that have been developed to integrate knowledge-based technology with conventional computing environments and the implications for both developers of knowledge-based applications and the end users.

In chapter 2 several methods of integrating knowledge-based systems with conventional-computing systems are presented followed by a survey of six commercial expert system tools. Chapter 3 discusses methods of integrating expert systems with relational database systems. Chapter 4 examines how external functions, written in conventional programming languages, can be called from an expert system in order to extend its computational capabilities. Chapter 5 explores the concept of embedding an expert system in a conventional system in order to achieve the tightest possible integration of both technologies. In chapter 6 the most flexible method of integrating expert system technology with conventional technology, a client-server architecture, is presented. Chapter 7 discusses knowledge management facilities in expert system shells. Chapter 8 summarizes all methods of integrating both technologies and lists all the features that an "ideal" expert system shell should have.

2. Integration Overview

Expert systems technology is best suited to solving problems that are very complex in nature, and generally involve uncertain facts and heuristic knowledge. On the other hand, tasks involving large amounts of data, complex mathematical calculations, graphical displays, and graphical user interfaces are better suited to conventional programming technology (e.g. database technology, C programming language, libraries of graphics functions). Real-world problems often involve both types of tasks. In order for expert system technology to be useful in solving problems it must be integrated with conventional programming technology. In recent years this requirement has been realized and various expert system packages have appeared on the market capable to varying degrees of integrating with conventional programming technology. Some of the methods used by those tools are discussed below.

2.1. Integrating Expert System Technology with Database Technology

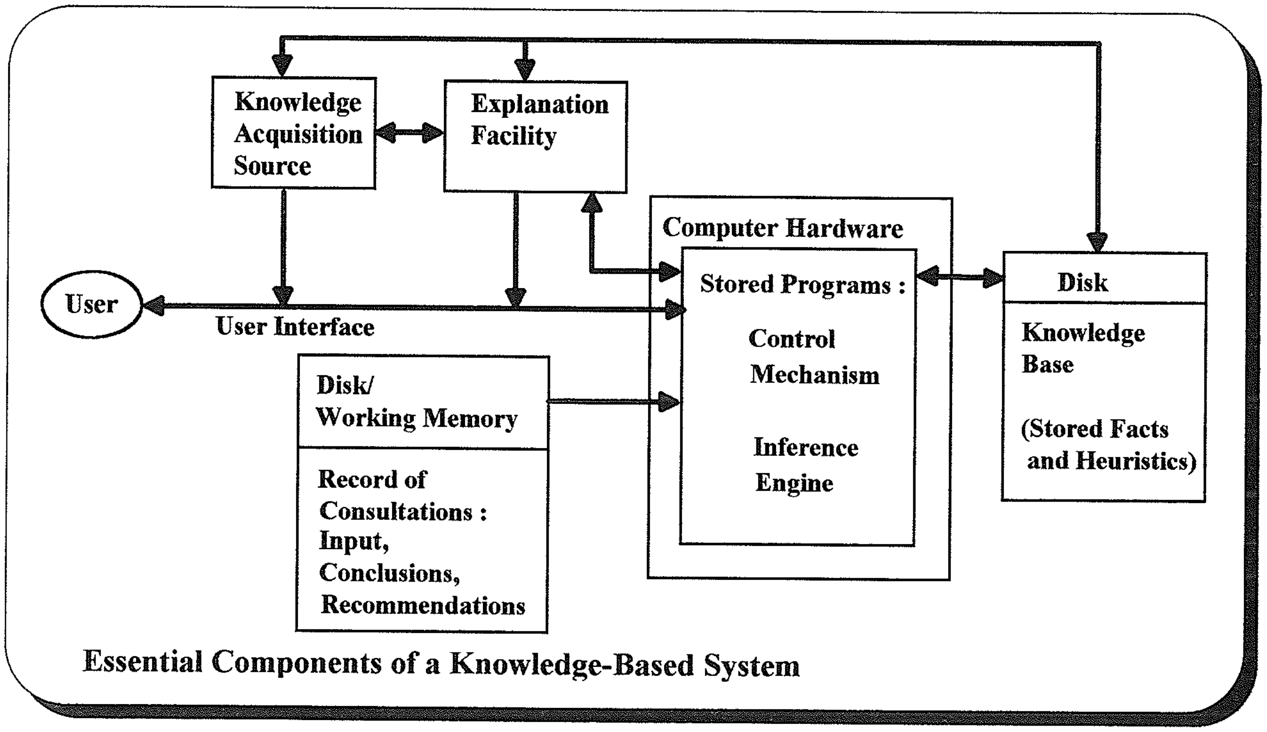

Knowledge Acquisition Source
Explanation Facility
Computer Hardware
Stored Programs :
Control Mechanism
Inference Engine
Disk
Knowledge Base
(Stored Facts and Heuristics)
User
User Interface
Disk/ Working Memory
Record of Consultations : Input, Conclusions, Recommendations
Essential Components of a Knowledge-Based System

Figure 2.1

Figure 2.1 shows all essential components of a knowledge-based system. The terms Knowledge-Based System and Expert System are often used as synonyms. This is not quite true since an Expert System is a Knowledge-Based System capable of solving very difficult problems requiring knowledge of a human expert. For the purpose of this thesis, however, both terms will be used interchangeably. The fundamental difference between a knowledge-based system and a conventional system is a separation of knowledge (stored in a knowledge base) and problem-solving logic (represented by an inference engine and a control strategy). A knowledge-based system starts processing by examining facts stored in working memory and matching the facts to the goal (if the goal is specified). Then it applies the knowledge stored in the knowledge base to the facts, according to a control strategy, which results in new facts being generated and stored in working memory. This process continues until either the goal is satisfied (a solution is found) or no new facts are generated [*Robert J. Mockler & D.G. Dologite*, 1992]. There are two basic methods for representing knowledge: rule-based and frame-based (object-oriented) representations.

A typical rule-based expert system consists of a knowledge base (represented as a set of rules) and a working memory which contains a set of facts. A rule-based system may use forward-chaining, backward-chaining, or a combination of both methods of reasoning (control strategy).

In a forward-chaining system (figure 2.2) the knowledge base is scanned for the rules that can be applied to the initial set of facts stored in working memory. Rules that can be applied to a given fact (antecedents of which match the fact) are put on an agenda as the rules that can potentially be "fired". The agenda is a prioritized queue and, depending on the implementation, developers can manipulate various aspects of the agenda (e.g. priorities of rules, depth-first or breadth-first rule ordering). The inference engine, after scanning the whole knowledge base, tests rule antecedents in the order of the agenda. If a rule evaluates to true it is "fired" and, as a result, a new fact concluded by the rule is added to the working memory. After testing all the rules from the agenda the inference engine repeats the cycle and looks for rules that can be applied to a newly generated facts. When there are no more rules that can be put on the agenda, processing is completed and the solution consists of all new facts generated by the "fired" rules [*Edmund C. Payne & Robert C. McArthur*, 1990].

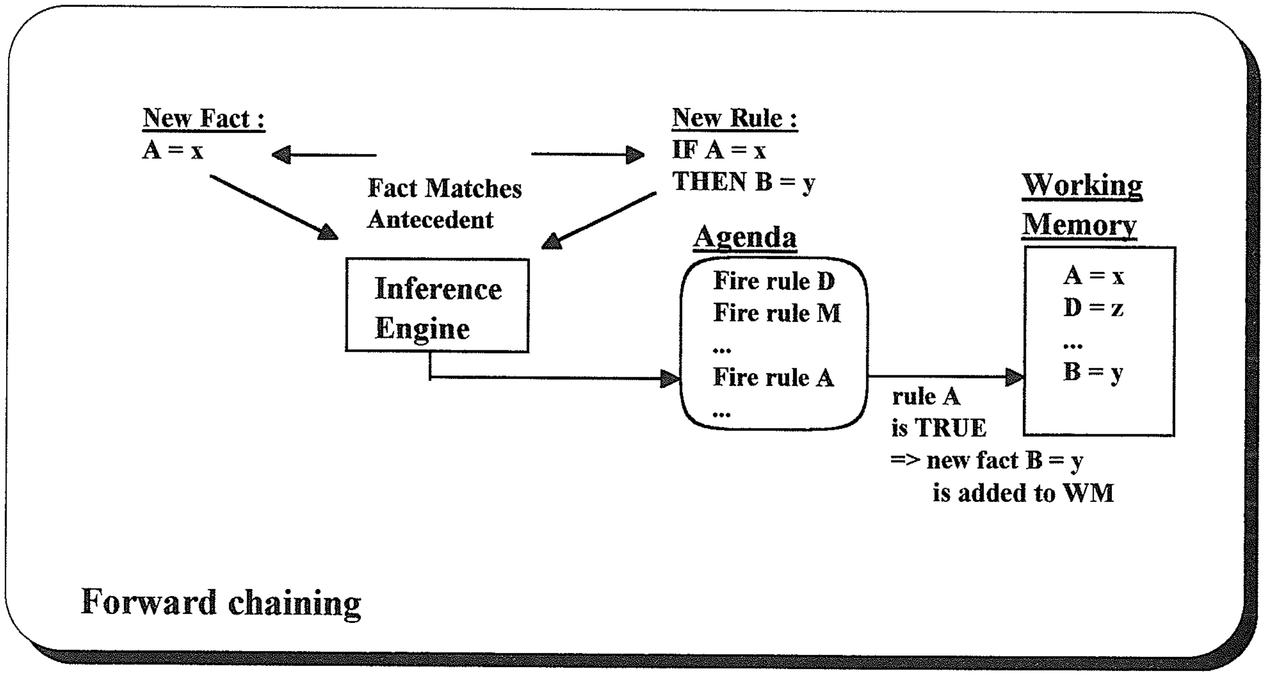


Figure 2.2

In a backward-chaining system (figure 2.3) the working memory is examined first. If any fact satisfies the goal the search for a solution is over. If no facts satisfy the goal the rules that conclude the goal are examined. If a rule concludes the goal its premise becomes a subgoal. Each subgoal can be satisfied either by finding a fact in the working memory or by finding a rule that concludes the subgoal. When all subgoals are satisfied the search for a solution is completed. The solution consists of all the rules that where successfully applied [*Edmund C. Payne & Robert C. McArthur*, 1990].

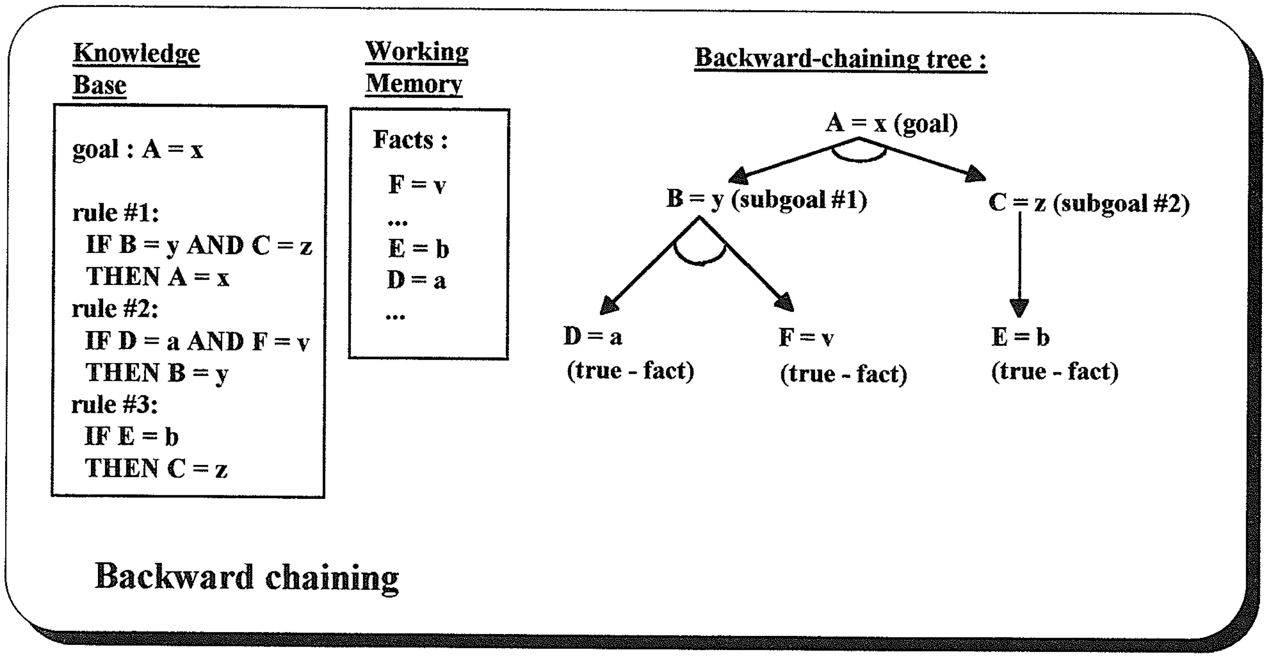


Figure 2.3

Both methods use initial facts as a way of describing the world's initial state of a particular problem that the system is to solve. First-Generation expert systems required the user to enter initial facts manually. This worked fine for simple systems with a few initial facts but as expert systems became more complex it did not suffice. Soon it became clear that if expert systems technology was to be used in solving real-world problems it had to be integrated with database technology. The first step in this direction was adding the ability to read data from a database into the working memory thus obtaining a set of initial facts. Typically the rules will access a database in the premise part and assign values to the facts in the conclusion part. The next step is storing the results of reasoning and/or intermediate facts in the database thus allowing other conventional or expert systems to do further processing based on the results obtained (figure 2.4).

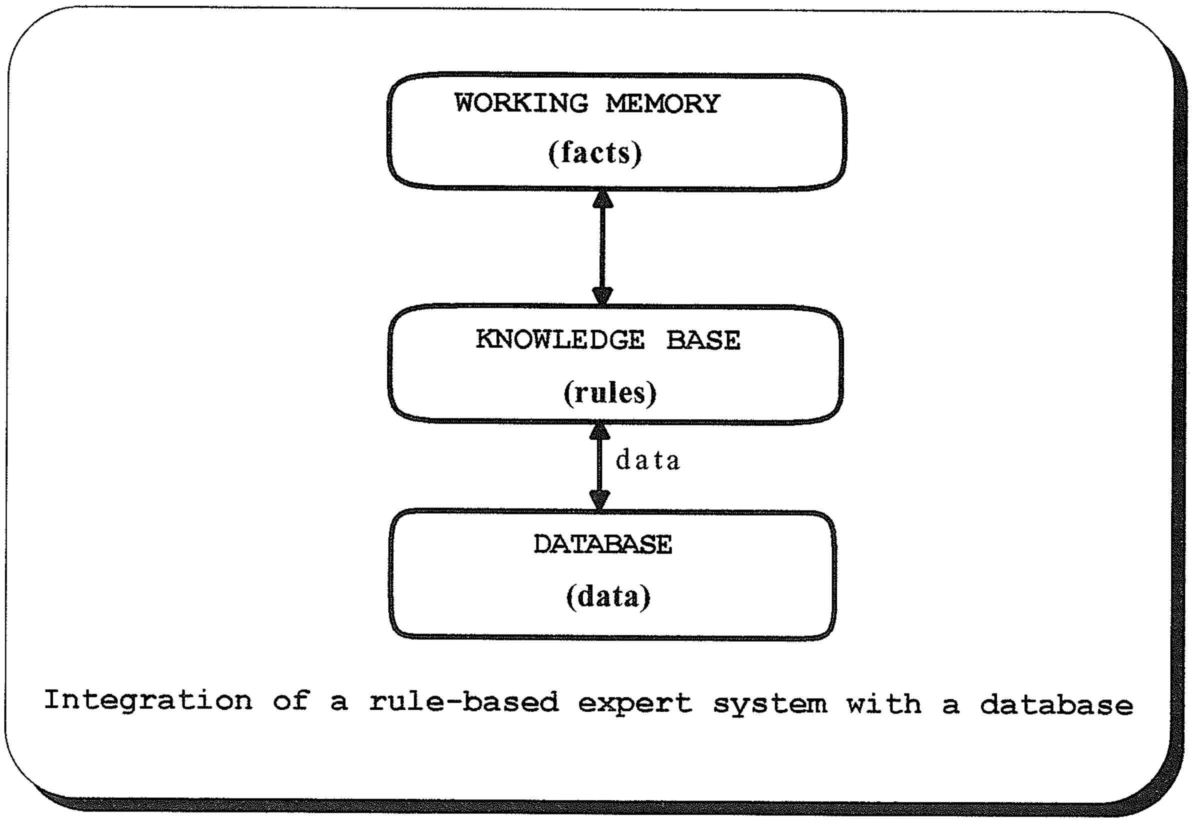


Figure 2.4

Another type of expert system development approach uses object-oriented (frame-based) techniques to represent knowledge (figure 2.5). Knowledge bases in such systems consists of a hierarchy of classes (or objects) some of which are derived from others. Subclasses inherit some properties from their parent classes and add new properties or over-ride some of their parents' properties. Some expert systems combine both rule-based and object-oriented knowledge representations. In an object-oriented system, values of attributes are determined either by inheritance or by using demons (when-needed methods and when-changed methods). These demons calculate values based on other known values, ask the user for values, or read values from the database (another implementation of integration of expert system technology with database technology - figure 2.6).

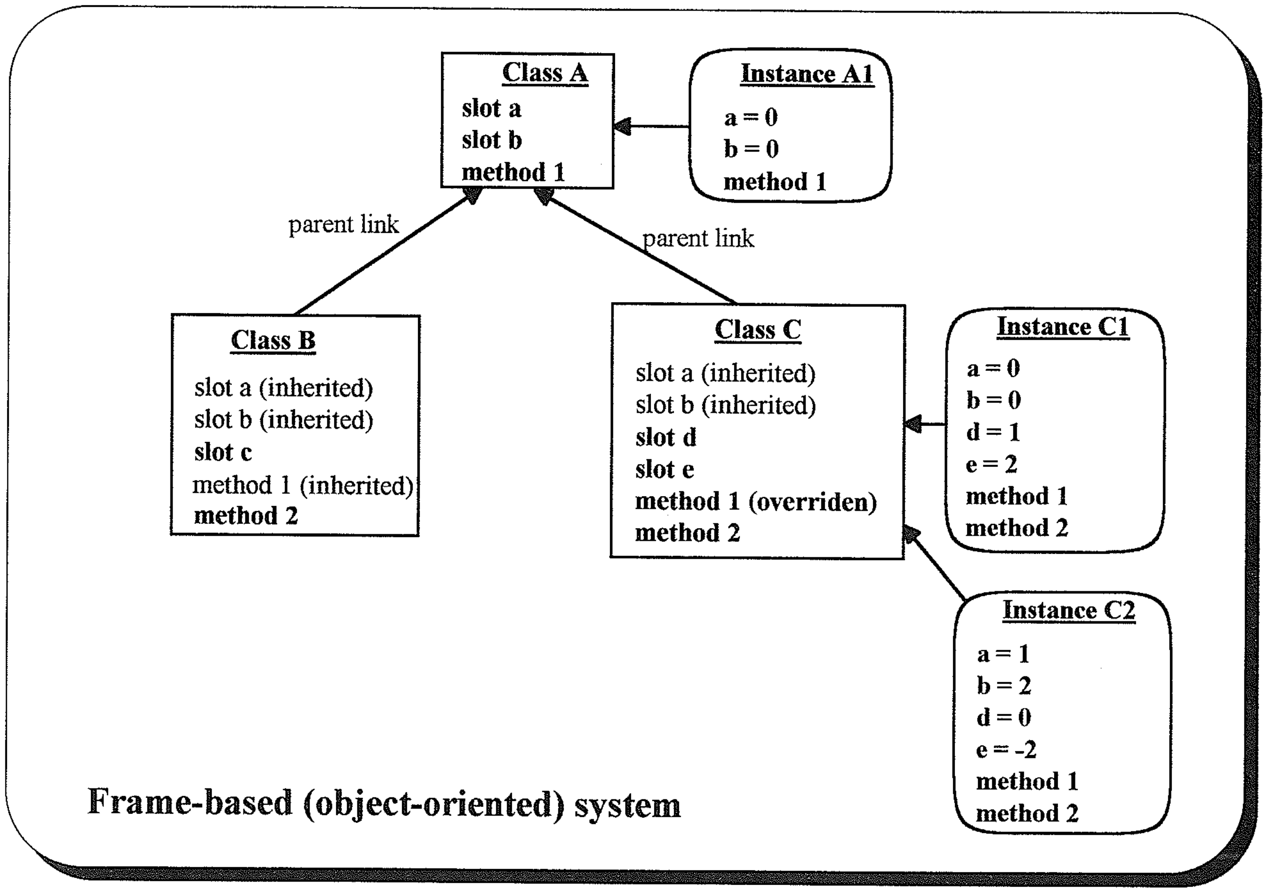
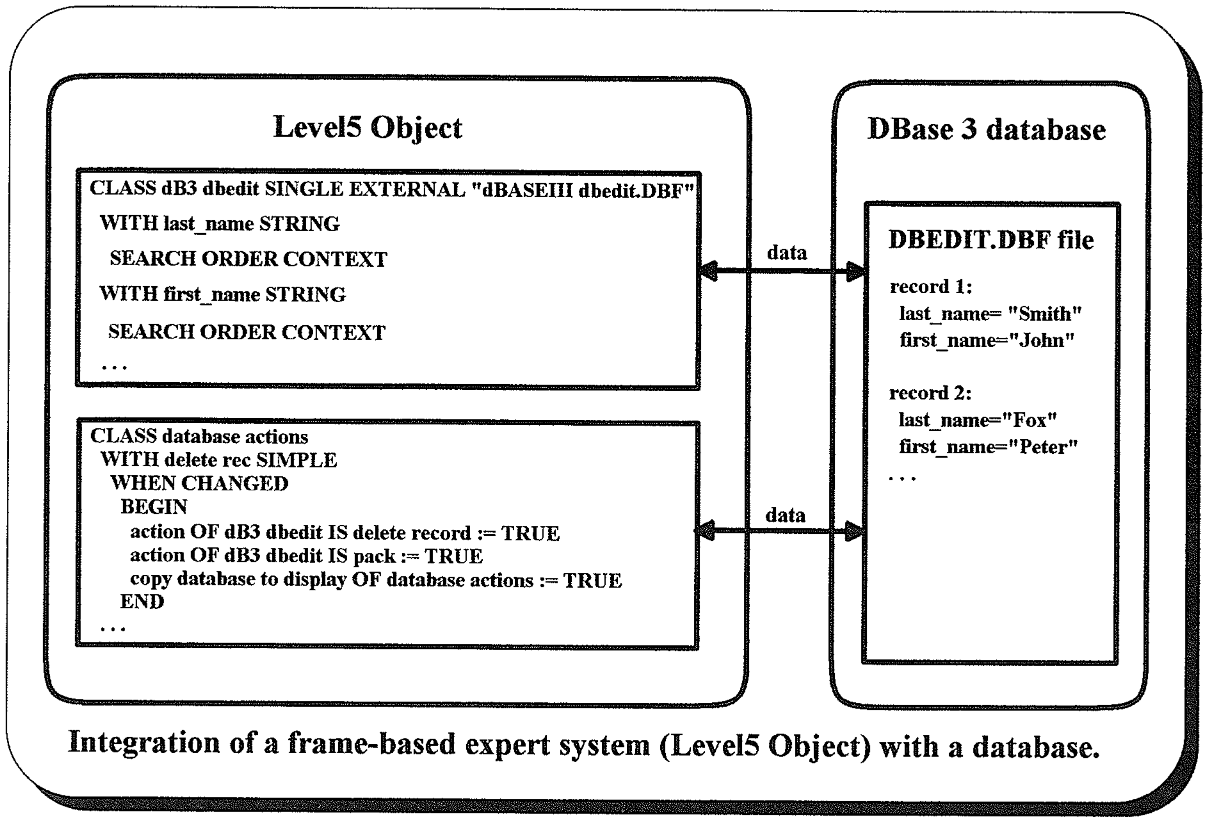


Figure 2.5

The advantages of integration of expert system technology with database technology are enormous. The expert system gains access to a huge amount of data describing the state of the world. It can analyze this data, create results, and store the results in a database so that other systems can process it further. Also, an expert system can take advantage of various features inherent in database technology such as concurrency, security, data consistency, and data query optimization [*Beynon-Davies, P.*, 1991]. Finally, expert systems can solve real world problems and can become a part of a larger system.



```
CLASS dB3 dbedit SINGLE EXTERNAL "dBASEIII dbedit.DBF"
 WITH last_name STRING
  SEARCH ORDER CONTEXT
 WITH first_name STRING
  SEARCH ORDER CONTEXT
 ...
```

```
CLASS database actions
 WITH delete rec SIMPLE
  WHEN CHANGED
   BEGIN
    action OF dB3 dbedit IS delete record := TRUE
    action OF dB3 dbedit IS pack := TRUE
    copy database to display OF database actions := TRUE
   END
 ...
```

Figure 2.6

The main disadvantage of integrating expert system technology with database technology is that the systems become more complex. Part of the knowledge base must be concerned with accessing a database, so the actual problem-solving knowledge may become less understandable and more difficult to maintain. Also depending on the implementation, the expert system may become less portable when it is coupled with a particular database system (e.g. DBase for IBM

PC computers). Typically such a database interface, implemented by the developer of an expert system shell, is not complete (e.g. it does not support record locking, indexes, multiple indexes, etc.). A better approach is to use a generic interface to a third-party database server using SQL. This can greatly increase an expert system's portability and flexibility [*Level5 Object - Rdb/SQL Interface Guide*, 1992]. Further discussion of these methods is presented in chapter 3.

2.2. Calling external programs/functions from a Knowledge-Based System

A knowledge-based approach to problem solving is appropriate if the problem to be solved is not well-defined, the knowledge is incomplete or uncertain, there is no clear algorithmic solution, or knowledge is heuristic in nature. On the other hand, tasks that have clear algorithmic solutions, that depend on mathematical calculations, that are connected with the user interface or graphics are better suited to conventional programming technology. Complex, real-world problems typically contain both types of tasks. Expert system tools typically are very limited in math and graphics functions and those functions that they do have are very inefficient when compared to conventional programming languages (e.g. C language). By allowing the expert system shell to call an external program or function, many of the above limitations can be eliminated.

Many second-generation rule-based systems support using external function calls in rule premises and conclusions (figures 2.7 and 2.8). Similarly, frame-based (object-oriented) systems now support calling external functions from within when-changed and when-needed methods.

Parameters to the functions and the results from the functions can be passed either through memory or ASCII files. Using ASCII files for communication can slow down the procedure significantly.

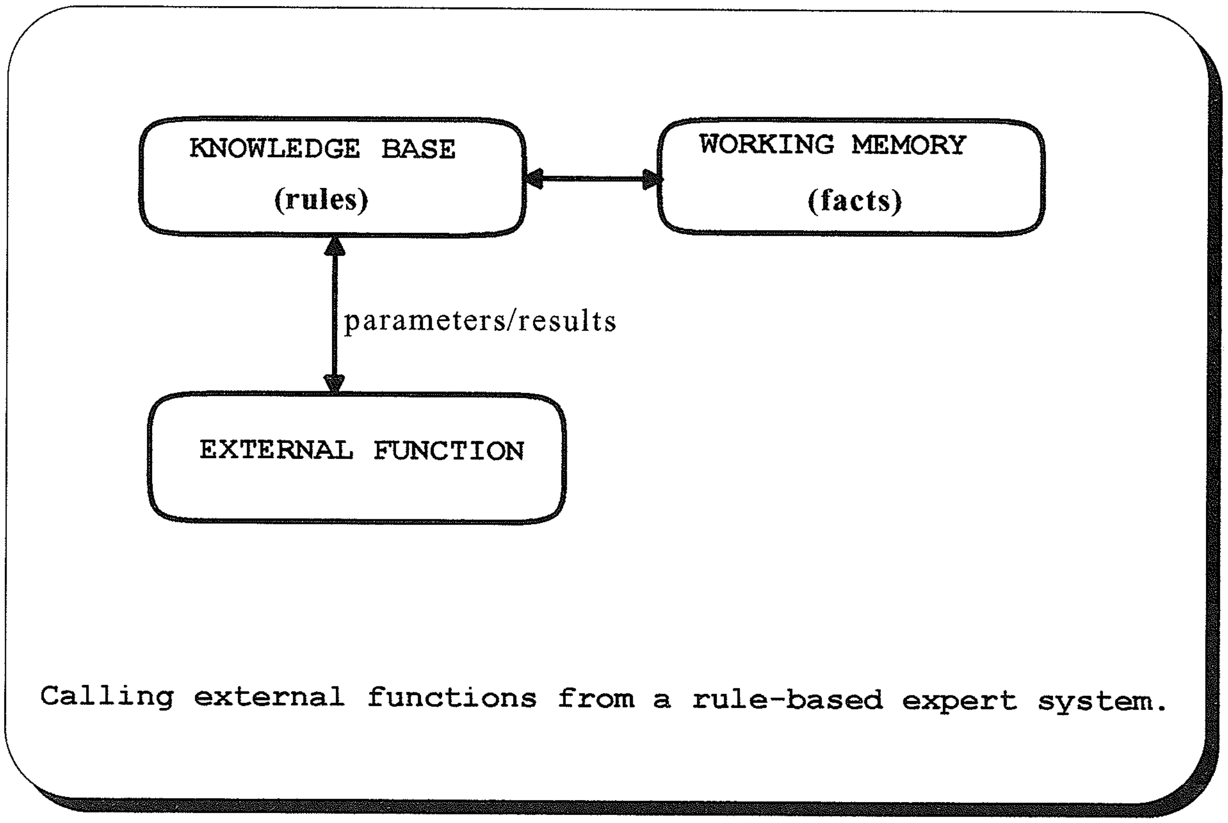


Figure 2.7

Allowing calls to external functions can significantly increase the capabilities and the performance of an expert system. There are no real disadvantages to this method assuming the implementation does not make it difficult or cumbersome to use as is the case in some expert system tools. Further discussion of these integration methods is presented in chapter 4.

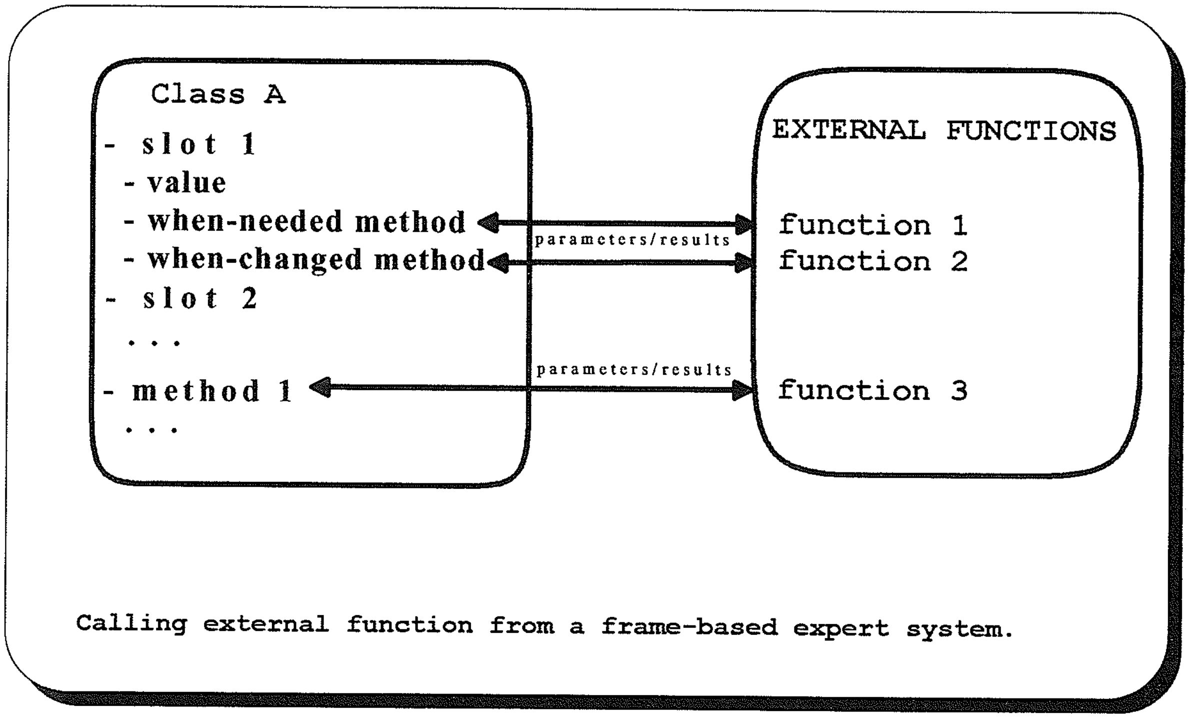


Figure 2.8

2.3. Embedded solutions

Embedding an expert system in a conventional system (or a conventional system in an expert system) is the tightest integration method (figure 2.9). It is not as flexible as client - server architecture but allows for closer and more efficient integration. Both integrated components can call each other's internal functions and access each other's data structures directly. In a typical scenario, an expert system is embedded in a conventional system. The conventional system provides the user interface, performs numerical calculations, displays the results, accesses a database, and calls the expert system component when necessary. Calling the expert system from the point of view of the conventional system is like calling any other function or procedure. After the expert system has completed processing the conventional system can read the results directly from the working memory of the expert system.

Advantages of an embedded solution are efficiency and very close integration. On the other hand this method is not very flexible - both components must be compiled together and no other external programs can be integrated with the system at runtime. Also the integration procedure may be quite complicated, it may require code modifications of some modules of both components. Further discussion of embedded solutions is presented in chapter 5.

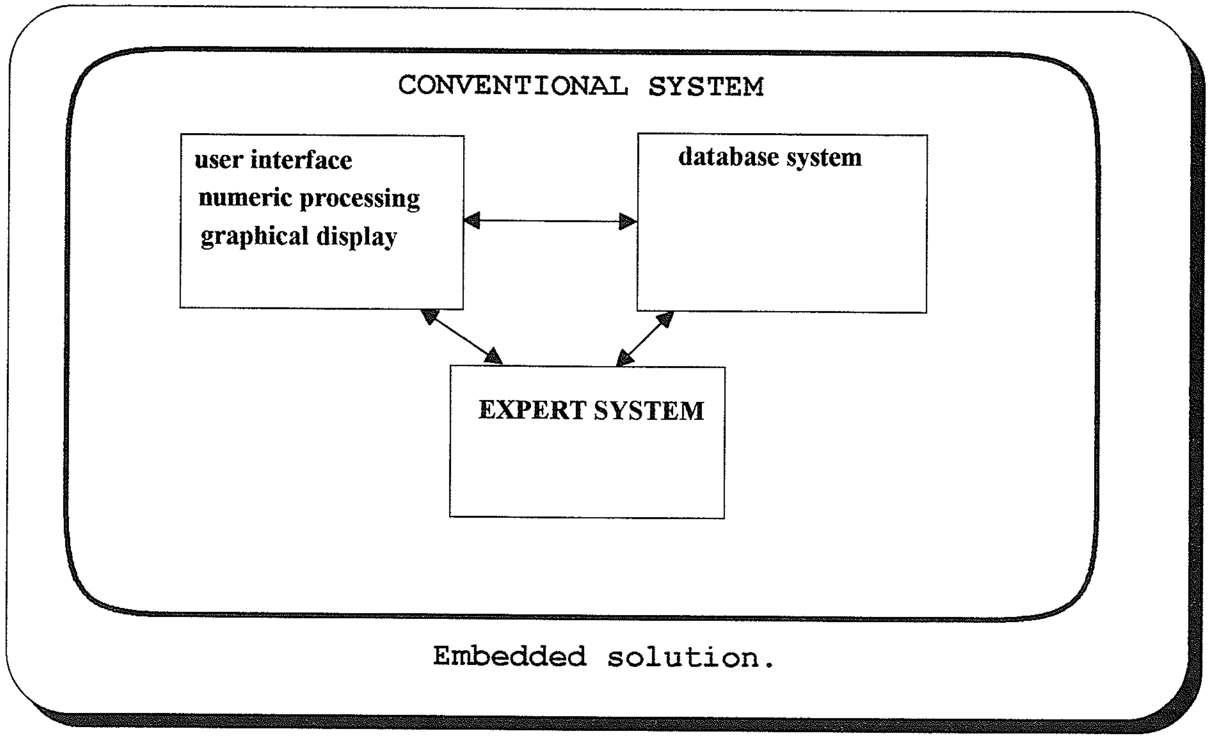


Figure 2.9

2.4. Client - Server Architecture

Another method of integrating expert system technology with conventional programming technology is through the use of client-server architecture (figure 2.10). This method can only be used in multitasking environments such as Microsoft Windows 3.x on IBM PC compatible computers. The client-server architecture is based on at least two programs running simultaneously. One of the programs is the expert system and the other is a conventional program (or another expert system). Both programs can communicate with each other. They can exchange data and they can request the other program to perform certain functions. Typically the program that initiates the conversation becomes a client (or a master) and the program responding to the request assumes the role of a server (or a slave). The roles can change dynamically over time. Also in a more complex scenario there can be more programs running simultaneously and carrying out conversations. Each program can be involved in multiple conversations at the same time and may assume the role of a client in some and a server in the others.

Client - server architectures are the most flexible method of integrating different technologies. The only requirement for the programs taking part in such a system is that they must all provide a common communication protocol. In the Microsoft Windows environment such a protocol called Dynamic Data Exchange (DDE) is already defined by the operating system. Most Windows-based applications support the DDE protocol. DDE allows both data exchange and sending commands. The only disadvantage of using client - server architecture is the overhead of establishing links between the programs and of the communication protocol itself. Also the programs must agree on what data and commands they want to exchange. Further discussion of client-server architectures is presented in chapter 6.

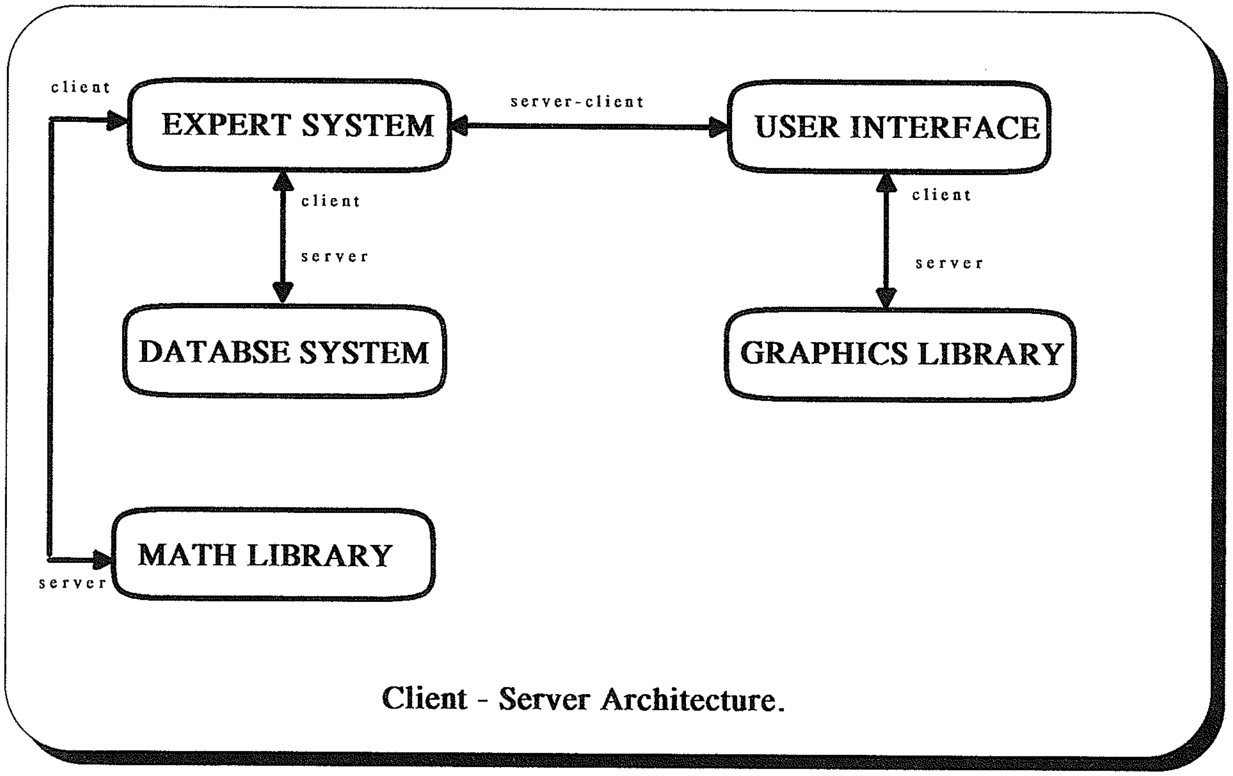


Figure 2.10

2.5 Ideal model of integration

According to some researchers (Earl D. Sacerdoti, 1989) an ideally integrated expert-system tool should have the following characteristics:

- be callable from C, COBOL, FORTRAN, and assembler;
- be able to call C, COBOL, FORTRAN, and assembler;
- be able to read from and write to relational databases via SQL;
- support well-documented conventions for communication with external applications;
- be configurable by developers;
- be compact enough to fit within a fraction of the directly addressable space of the chosen hardware;
- create reentrant executable versions of the system once the development features have been stripped out;
- be available on a wide range of systems.

The four methods of integration discussed above come very close to this ideal. There is no expert system tool currently on the market that would satisfy all the above requirements but some tools are evolving in that direction.

2.6 Survey of six commercial expert system tools

Implementation of the methods of integrating expert system technology with conventional technology described above is different in various tools currently available in the market. Level5 Object from Information Builders Inc. will be the primary tool used for illustrating the issues involved in integration of expert system technology with conventional technology. Various other expert system tools will be briefly examined to show alternative approaches.

Six expert system tools - ART-IM, CLIPS, KES, Level 5, VAX OPS5, and Kappa-PC - will be introduced in the next section. Their functionality, performance, advantages, and disadvantages will be briefly described [based on *William Mettrey*, 1991]. Each of these systems will be used in subsequent chapters to illustrate various ways of providing integrated expert system facilities.

2.6.1. C Language Integrated Production System (CLIPS)

CLIPS was developed by NASA at the Lyndon B. Johnson Space Center. It was designed to overcome a number of difficulties NASA had experienced using Lisp-based tools, including low availability of Lisp on conventional computers, high cost of Lisp-based tools and hardware, and poor integration of Lisp with other languages [*William Mettrey*, 1991]. CLIPS is written in C to support the goals of high portability, low cost, and ease of integration with external systems. CLIPS was designed as a rule-based system based on the architecture of ART - NASA's Lisp-based tool. According to NASA, CLIPS has been delivered to more than 2,500 users, and is available on all hardware platforms.

CLIPS uses rules as its primary knowledge representation approach. It uses a Lisp-like rule syntax:

```
(defrule  Rule-Name
    "Optional Documentation String"
    (condition-1)      ; The left-hand side is composed of
    (condition-2)      ; zero or more conditions
    (condition-n)      ; each enclosed in parentheses
=>  (action-1)         ;The right-hand side is composed of
    (action-2)         ; zero or more actions
    (action-n) )
```

CLIPS supports a rich pattern-matching language for specifying rule conditions which operates on both single fields and multifield sequences composed of strings, symbols, and numbers. CLIPS also supports templates as a means of specifying rule conditions. In addition, CLIPS provides procedural programming constructs (if...then...else, while) on the right-hand side of rules. The above features enable CLIPS to express in a single rule, knowledge that requires several rules in other expert system tools. In version 5.0 CLIPS Object Oriented Language (COOL) was introduced. It extends CLIPS's capabilities by supporting object-oriented programming [*Tom Brooke*, 1992].

CLIPS inference mechanism is based on a forward-chaining control strategy that implements the classic recognize - act cycle. Conditions of rules are matched with facts in the knowledge base. Rules with all conditions satisfied are instantiated (activated) and placed on an agenda (or in a conflict set). CLIPS selects the rules with the highest salience (priority), which can vary from -10,000 to +10,000 (the default is 0), to fire. Firing a rule consists of performing the rule's actions (specified on the right-hand side). Forward chaining is implemented using an efficient Rete matching algorithm. The Rete algorithm uses a network representation of rules dependencies and its major advantage is a very fast evaluation of rules premises [*C. Forgy*, 1982]. CLIPS does not support backward chaining [*William Mettrey*, 1987].

CLIPS is offered on a wide range of hardware platforms. Its strengths include strong support of forward chaining, ease of integration with external systems, portability, fast execution, and low price. Its main weakness is its lack of support for backward chaining. Templates provide the structuring capabilities of a frame system, but do not support inheritance or procedural attachments. This has been corrected in version 5.0 by introducing COOL (CLIPS Object Oriented Language) which allows object-oriented programming, although COOL has not been completely integrated with CLIPS as yet [*Tom Brooke*, 1992].

2.6.2. Automated Reasoning Tool for Information Management (ART-IM)

ART was introduced in 1985 by Inference Corporation as a Lisp-based expert system tool targeted to Lisp-machines and high-end workstations. ART-IM was developed using NASA's CLIPS as a base and adding several enhancements, most important of which was a schema (frame) system and an object-oriented programming capability.

From the knowledge representation perspective, the main difference between ART-IM and CLIPS is the frame system. ART-IM refers to frames as schemata, which can be used on the left-hand side of rules. A schema consists of a schema name and one or more slots. The slots represent either attributes of a schema or its relationship with other schemata. A schema can be defined either statically using the *defschema* statement or dynamically at run-time. ART-IM supports single inheritance - values and functions are inherited via *is-a* and *instance-of* relations between schemata e.g.:

```
(defschema                 machine-1
  (instance-of             machine)
  (machine-status          idle)
  (current-part            P9)
  (current-operation       OP-3))
```

The inference mechanisms of ART-IM and CLIPS are very similar, with forward chaining being the primary mechanism. In addition, ART-IM provides object-oriented programming capabilities. An ART-IM object is represented by a schema whose slots contain values for the object's attributes and functions to carry out the object's actions. Functions can be written in C or using ART-IM commands.

ART-IM has all the advantages of CLIPS. Its frame system and object-oriented programming capability make ART-IM even more powerful and flexible, increasing the range of possible applications. ART-IM is an expensive tool targeted at the high-end market. It offers a very comprehensive debugging and development environment including a windowed user interface. Numerous applications have been developed using ART-IM and it is capable of handling very large knowledge bases [*William Mettrey*, 1991]. Some of the weaknesses of ART-IM are lack of support of backward chaining, multiple inheritance, and user-defined inheritance.

2.6.3. Knowledge Engineering System (KES)

KES was introduced by Software Architecture & Engineering in 1982. The early versions of KES were implemented in Lisp but it was ported to C in version 2.1. KES historically consisted of three subsystems: KES Bayes, KES HT, and KES PS. KES Bayes is a statistical pattern classification subsystem for applications that have a large body of data expressed as probabilities. This subsystem is no longer supported. KES HT is a hypothesis-and-test subsystem that is useful for specialized diagnostic applications. KES PS, the production system module, is the most frequently used of KES's subsystems. KES PS will be further referred to as KES.

KES provides forward-chaining rules (demons), backward-chaining rules, and a class (frame) system for knowledge representation. The KES equivalents of facts are called attributes. KES has a rigid typing system - each attribute must be declared and given a type which specifies the kind of values it can assume and the operations that can be performed on it. KES can also handle uncertain knowledge by using certainty factors[1]. The KES equivalent of a frame is a class. Classes can be used to specify attributes tested by rule conditions. Classes support single inheritance but do not allow procedural attachments. Intrinsic and user-defined functions, however, can be called from both rules and demons. The general form of a KES forward-chaining rule is the following:

```
rule name:
    variable declarations    \Optional variable declarations
when                         \Keyword when signals the start of the left-hand side
    condition(s)             \One or more LHS conditions
then                         \Keyword then signals the start of the right-hand side
    action(s)                \One or more RHS actions
endwhen.                     \Keyword endwhen terminates the rule
```

The general form of a KES backward-chaining rule is the following:

```
if                           \Keyword if signals the start of the left-hand side
    antecedent(s)            \One or more antecedents
then                         \Keyword then signals the start of the right-hand side
    consequent(s)            \One or more consequents
endif.                       \Keyword endif terminates the rule
```

[1] Certainty Factors (CFs) are numeric values (typically in the range from 0 to 1) that represent a degree of certainty (or uncertainty) of facts and/or rules in a knowledge-based system. Special rules have been defined to allow combining two or more CFs, for example if a rule uses, in its premise, fact A with CF=0.5 and fact B with CF=0.5 then the conclusion of this rule will have a CF=0.25.

KES's primary inference mechanism is backward chaining. Forward-chaining rules (demons) cannot interfere with backward-chaining rules - a demon cannot contribute a value to an attribute that is being pursued by a backward-chaining inference. KES does not use the Rete algorithm in forward chaining so its forward chaining processing is not very efficient. Also, KES performs a depth-first evaluation of the forward-chaining rule conditions that can lead to firings of unwanted rules or multiple firings of the same rule [*William Mettrey*, 1991].

KES supports all major knowledge representation mechanisms. KES executes on a number of hardware platforms, including IBM mainframes, IBM PC's, DEC VAXs, and other workstations. An application developed in KES can be embedded in external user code that is written in C. The weak element of KES is its forward chaining mechanism - caution has to be exercised to avoid undesirable rule firings.

2.6.4. VAX Official Production System Version 5 (VAX OPS5)

VAX OPS5 is a descendant of several production system languages developed at Carnegie Mellon University. It is written in Bliss (DEC's system implementation language) and executes only on DEC hardware.

Knowledge is represented in VAX OPS5 by rules which have a Lisp-like syntax (similar to that of CLIPS and ART-IM). The left-hand side conditions are composed of attribute-value pairs. Values can be either symbols or numbers; strings are not supported. Pattern-matching functionality is not as rich as that in CLIPS or ART-IM. VAX OPS5 also does not support procedural programming constructs. The general form of a VAX OPS5 rule is the following:

```
(PRule-Name
  (condition-1)     ;The left-hand side is composed of
  (condition-2)     ;one or more conditions
  (condition-n)
==>
  (action-1)        ; The right-hand side is composed of
  (action-2)        ;one or more actions
  (action-n) )
```

VAX OPS5's only inference mechanism is forward chaining. It is based on an efficient Rete algorithm. VAX OPS5 provides two conflict resolution strategies, Lexicographic Sort (Lex) and Means-Ends Analysis (MEA), for selecting rules to fire.

VAX OPS5 strengths are rapid execution times, integration with other DEC software, and the proven ability to support the development and delivery of large expert systems (Xcon, Xsel). Among the weaknesses of VAX OPS5 are lack of support for backward chaining, and frames, and non-portability to other hardware platforms [*William Mettrey*, 1991].

2.6.5. Kappa-PC

Kappa-PC is a Microsoft Windows-based expert system shell developed by IntelliCorp, a company known for its Lisp machine-based expert system tool - KEE. Kappa-PC's knowledge representation and inference mechanisms are directly related to those of KEE.

Kappa-PC supports both frame-based (object-oriented) and rule-based knowledge representation. *Classes* and *instances* are used to represent objects. Classes can contain both *slots* and *methods*. Inheritance is supported by the means of *subclasses.* A *subclass* can inherit both *slots* and *methods* from its *ancestor*. *Methods* are written in KAL (Kappa-PC's Application Language) which provides a rich set of built-in functions, operators, knowledge representation and manipulation constructs, and procedural-programming structures (e.g. *For, ForAll, While).* Also users can write their own functions in KAL. Kappa-PC provides a special type of methods called *monitors. Monitors* are methods that are linked to slots and are triggered either by changes in the slot value or by a request for a slot value that is not known. There are four types of *monitors*: *If Needed, When Accessed, Before Change,* and *After Change*. Both *classes* and *instances* can be declared statically or created dynamically at run-time, for example:

```
For counter [1 10 0.5]
   {
     MakeInstance(Obj#10*counter, Root);
     MakeSlot(Obj#10*counter, Size);
     SetValue(Obj#10*counter, Size, counter);
   };
```

Beside an object-oriented system, Kappa-PC supports a rule-based knowledge representation. Both forward and backward chaining inference mechanisms are supported. Kappa-PC allows the developer to select one of the four *Conflict Resolution Strategies* during forward chaining: *Selective* (follows only one successful path), *Depthfirst* (exhaustive search, new facts added to the top of the Agenda), *Breadthfirst* (exhaustive search, new facts added to the end of the Agenda), *Bestfirst* (exhaustive search, rules are selected in their Priority order). Both forward and backward chaining rules use the same syntax e.g.:

```
SluggishTurnover:
[car:Autos]
IF car:IgnitionKey #= ON And
   car:ElectricalSystem #= Bad;
THEN car:EngineTurnover = Sluggish;
```

Kappa-PC is a very powerful expert system tool targeted for the high-end PC user market. In addition to rich knowledge representation and flexible inference mechanism, Kappa-PC provides a wide set of development-support tools, including a debugger, user interface functionality, interfaces to external functions, databases and DDE (Dynamic Data Exchange). Despite some minor deficiencies (e.g. poorly designed explanation facility) Kappa-PC is one of the most powerful expert system tools currently on the market [*Kappa-PC User's Guide*, 1992].

2.6.6. Level5 Object

Level5 Object evaluated below (and in the remaining chapters) is a Microsoft Windows-based expert system shell which allows both rule-based and frame-based (object-oriented) knowledge representation.

The knowledge base in Level5 Object is built around classes which consist of attributes. Classes can inherit attributes from other classes. Each attribute of a class can have when-needed and when-changed methods attached (similar to monitors in Kappa-PC). Classes are instantiated by objects. One class can be instantiated by many objects. Attributes of objects can be assigned values that can be later accessed in when-needed and when-changed methods as well as in rules. Part of knowledge in Level5 Object can be represented in rules. There are two types of rules: demons (forward-chaining rules) and rules (backward-chaining rules). Rules can use attributes of objects both in premises and in conclusions, for example:

```
IF last_name OF employee = "Smith"
AND first_name of employee = "John"
THEN delete OF actions = TRUE
```

In addition to knowledge representation and processing mechanisms described above, Level5 Object is equipped in a variety of tools allowing to quickly create very attractive and flexible user interface. It also contains several means of integration with conventional systems (these will be discussed in detail in the following chapters): access mechanisms to external databases, mechanism for calling external programs and support for Dynamic Data Exchange. Level5 Object can also handle uncertain knowledge using certainty factors [*Level5 Object User's Guide*, 1990].

Level5 Object is a very good prototyping tool. Its extensive support for user interface allows quick development of attractive prototype systems. Its support for all three major knowledge representation schemes (forward and backward chaining rules, classes) makes Level5 Object applicable to a wide range of problems. Its English-like rule syntax is easy to learn and understand, although its pattern-matching functionality is poor when compared to other tools. Level5 Object is also available on the IBM mainframe and Unix hardware platforms.

2.7. Summary

AI researchers have finally realized the importance of integrating expert system technology, as well as other AI technologies, with conventional-computing environments. Many commercial expert system shells now support various methods of this integration but none provides full integration. As discussed earlier, there are four major methods of integrating expert system technology with conventional-computing systems: 1) an interface to external databases; 2) an interface to external functions written in conventional-programming languages; 3) embedding expert systems in conventional systems; and 4) client-server architectures. All commercial tools described in chapter 2 use one or more of these methods but their implementations usually impose various limitations and are not complete. For example, Level5 Object supports a client-server architecture (described in detail in chapter 6) through the DDE protocol, but the implementation is only partial - Level5 Object cannot act as a server. Despite such limitations most tools evolve in the right direction and in the near future we can expect the arrival of expert system tools able to fully integrate with conventional-computing environments.

The following chapters discuss methods of integrating expert system technology with conventional technology. For the purpose of this thesis conventional technology means computer hardware and software technology developed outside of Artificial Intelligence. Conventional hardware means general-purpose computer architectures, like mainframe computers, Unix-based workstations, and microcomputers, as opposed to Lisp-machines and neural network circuits. Conventional software includes database systems and other applications that can solve only very specific problems and work according to a pre-designed algorithms. Conventional applications are usually developed using procedural programming languages like Pascal, the C language, P/L1, or Fortran.

3. Integrating Expert System Technology with Database Technology - Implementation

Integration of expert system technology with database technology makes expert systems much more powerful and useful. It gives them access to enormous amounts of data that can be used by expert systems in their reasoning as a source of facts representing the state of the real world. It also enables expert systems to store the results of their analysis in a database that can be processed further by other applications. As a mature technology, with integrity control and concurrency mechanisms, database technology also improves the quality and reliability of expert systems [*Jarke, M. and Y. Vassiliou,* 1984].

The need for access to external data in expert system tools was realized quite early in their development. The first step addressing this need was the ability of expert system tools to read from and to write to ASCII files. This was a solution for very simple systems only. The second step in integration of commercial expert system tools with conventional technology was a database interface.

Various commercial expert system tools implement integration with database technology differently, but at least the following features should be present regardless of the implementation:

- ability to both read and write database records;
- ability to use indexes to access records;
- ability to append and delete records;
- ability to process records both sequentially and randomly;
- ability to search for a particular record;
- ability to detect the end-of-file condition.

Most expert system tools available in the market support at least one database system (e.g. DBase on IBM PC platform). Some tools support more than one database system (e.g. DBase and Focus are supported by Level5 Object). The most recent trend however is using a database gateway in which an interface, typically in the form of SQL[1], to a third-party database system is used. This approach allows a developer of an expert system tool to concentrate on knowledge-related aspects of the tool and it makes a database interface more generic, more flexible, more reliable, and more portable [*Level5 Object Rdb/SQL Interface Guide*, 1992].

3.1. Database technology in Level5 Object

The latest version of Level5 Object supports two database systems: DBase III and Focus [*Level5 Object User's Guide*, 1990]. It also offers a more generic database interface through SQL and third-party subsystems (DEC Pathworks for DOS, and VAX-based Rdb database system). Still another, more flexible, database interface provided by Level5 Object utilizes DDE (Dynamic Data Exchange - described in detail in chapter 6) to transfer SQL statements to a third-party relational database.

3.1.1. DBase interface in Level5 Object

Level5 Object implements its DBase interface using an object-oriented approach. The foundation of this system is the dB3 system class[2] which has the following structure:

---

[1] **Structured Query Language** (SQL) is a standard interface to relational database systems.

[2] **System classes** in Level5 Object are built-in object classes with associated functionality. System classes cannot be modified but a user can create his own classes that inherit properties from the system classes.

```
CLASS[1] dB3
WITH access COMPOUND[2]
    read,
    write,
    read shared,
    write shared
WITH action COMPOUND
    advance,
    previous,
    top,
    bottom,
    append record,
    insert record,
    delete record,
    recall record,
    close,
    open,
    pack
WITH eof SIMPLE[3]
WITH record NUMERIC
WITH size NUMERIC
WITH index file STRING
WITH filename STRING
WITH active SIMPLE
```

The **access** attribute of the dB3 class defines the database access privilege. It can be one of four:

-read	read-only exclusive access;
-write	both read and write exclusive access (default setting);
-read shared	read-only shared access;
-write shared	both read and write shared access.

The **action** attribute allows performing various actions on a database file. The **action** can assume the following values:

- open	a database file is opened;
- close	a database file is closed;
- advance	a current record pointer is moved to the next record;
- previous	a current record pointer is moved to the previous record;
- top	a current record pointer is moved to the first record;
- bottom	a current record pointer is moved to the last record;

[1] A **CLASS** in Level5 is very similar to a class in the C++ programming language - it consists of attributes, which in turn can have methods (procedures) attached to them.

[2] **COMPOUND** attribute type is like Enumerated type in Pascal, for example the **access** attribute can only assume one of the following values: **read**, **write**, **read shared**, and **write shared**.

[3] **SIMPLE** attribute type corresponds to Boolean in Pascal.

- append record	a new record is appended to the end of file;
- insert record	a new record is inserted at the current record position;
- delete record	a current record is marked as deleted;
- pack	all records marked as deleted are removed from a file;
- recall record	a record marked as deleted becomes undeleted.

The remaining attributes allow testing and controlling various aspects of a database:

- active	if TRUE a current record is active;
- record	it contains a current record number;
- size	it contains the number of all records in a database file;
- eof	if TRUE the end of file has been reached;
- file name	it contains the disk file name (must be specified before a file can be opened);
- index file	if specified the index file with this name will be used in the following operations: top, bottom, previous, advance, and FIND.

In order to use a database file, a class corresponding to that file must be created. All attributes of that class, both attribute names and types, must correspond to the fields of the DBase file. Level5 Object can create a class corresponding to a given DBase file automatically. The class will inherit all attributes of the dB3 system class described above. Lets consider an example EMPLOYEE database file with the following fields:

- NO	Numeric: employee number
- NAME	Character: employee name
- BORN	Date: employee's date of birth
- MALE	Logical: if TRUE then male if FALSE then female
- SALARY	Numeric: employee's salary

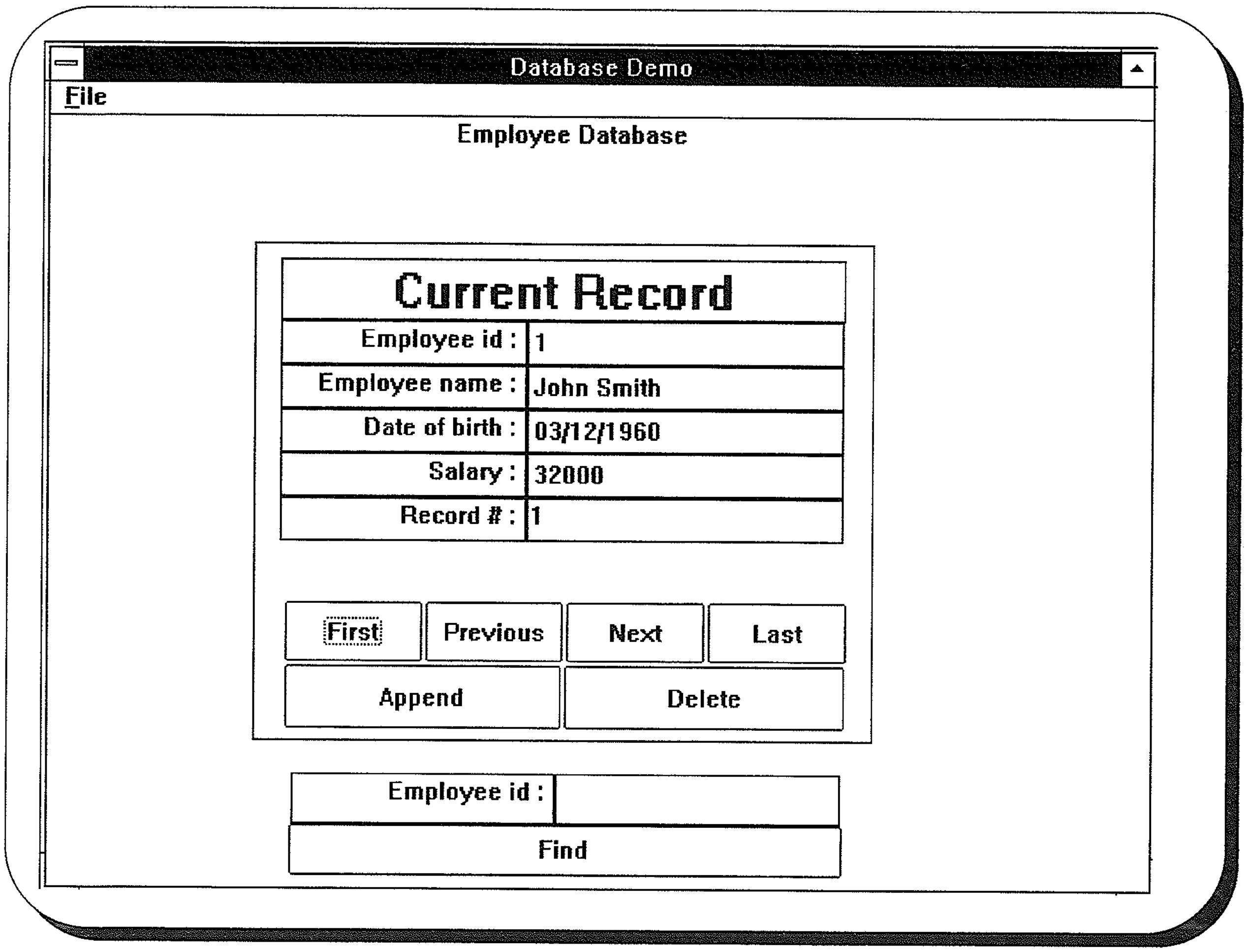
A dB3Employee class corresponding to the above DBase file would inherit all attributes of the dB3 system class plus it would have the following attributes:

- no	NUMERIC
- name	STRING
- born	TIME
- male	SIMPLE
- salary	NUMERIC

The attributes of the dB3-derived classes can be used both in rules and in methods within Level5 Object. The following examples use rules but the syntax is the same for methods (when-needed and when-changed):

```
RULE for opening a database file
IF fileOpen = TRUE
THEN open OF dB3Employee := TRUE

RULE for closing a database file
IF fileClose = TRUE
THEN close OF dB3Employee := TRUE

RULE for advancing to the next record
IF nextRecord = TRUE
THEN action OF dB3Employee IS advance

RULE for searching for a particular record and updating it if found
IF name OF dB3Employee <> "Brian Fox"
AND eof OF dB3Employee = FALSE
THEN action OF dB3Employee IS advance
AND LOOP
ELSE salary OF dB3Employee := 30000


RULE for adding a new record
IF appendRecord = TRUE
THEN action OF dB3Employee IS append record
AND no OF dB3Employee := LastNo + 1
AND name OF dB3Employee := "Marry Jones"

RULE for deleting and removing a record from a file
IF deleteRecord = TRUE
THEN action OF dB3Employee IS delete record
AND action OF dB3Employee IS pack

RULE for searching for a particular record (fast if index file is used)
FIND dB3Employee WHERE name OF dB3Employee = "John Black"
LIMIT 1
WHEN FOUND
recFound := TRUE
WHEN NONE FOUND
recFound := FALSE
FIND END
THEN searchedForName := TRUE
```

The attributes used in the above rules can be linked to buttons of the display as shown on Fig. 3.0. For example, the attribute *nextRecord* can be linked to the *Next* button and the attribute *deleteRecord* can be linked to the *Delete* button. Clicking with a mouse on a button will set the corresponding attribute value to TRUE.



Employee Database

Current Record

Employee id :	1
Employee name :	John Smith
Date of birth :	03/12/1960
Salary :	32000
Record # :	1

Employee id :	

Fig. 3.0

3.1.2. SQL Interface to a Remote Database in Level5 Object

Level5 Object provides an SQL interface to remote Rdb databases running on a VAX [*Level5 Object Rdb/SQL Interface Guide*, 1992]. Figure 3.1 shows the remote database query process:

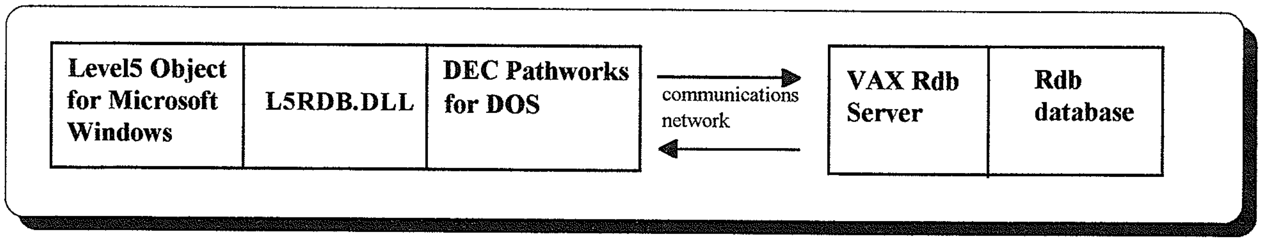


Figure 3.1

Level5 Object provides the L5RDB.DLL library which processes SQL queries and sends them to DEC Pathworks - a third-party software package. DEC Pathworks handles further communication with a remote VAX Rdb server that controls access to VAX-based Rdb databases. Responses to SQL queries are sent back the same way to Level5 Object. Level5 Object provides a new *EXEC SQL* command which triggers the processing described above, for example:

```
WHEN CHANGED
BEGIN
  EXEC SQL select model, make, MSRP from car
    where country = "Japan"
  END SQL INTO selected cars
END
```

The above when-changed method could be attached to an attribute that is linked to a *Japan Cars* button for example. Clicking on the *Japan Cars* button would trigger the when-changed method that would send an SQL query to a VAX-based database. The query would select records, containing *model* and *make* columns (attributes), from the *car* table (database file). Only records with *country* = "Japan" would be selected. The resulting table would be sent back to Level5 Object. Level5 Object would then create an instance of *selected cars* class for each row (record) of the result table and set the corresponding attributes to the values of *make* and *model*.

Level5 Object supports the following SQL commands:

- SELECT: specifies the data to be retrieved from a database and creates a result table;
- INSERT: inserts values as new records in a table;
- UPDATE: modifies rows in a table;
- DELETE: deletes rows from a table;
- CREATE: creates a new table;
- ALTER: changes an existing table;
- DROP: deletes specified SQL elements such as INDEX, SCHEMA, TABLE;
- DECLARE SCHEMA: specifies the name and source of the schema definitions to be accessed;
- DECLARE TRANSACTION: specifies a transaction;
- SET TRANSACTION: begins a transaction;
- COMMIT: makes permanent any changes made during a transaction;
- ROLLBACK: undoes any changes made during a transaction;
- GRANT: creates privileges to the access control list;
- REVOKE: removes privileges to the access control list;
- COMMENT ON: modifies or adds a comment about a table.

When ***SQL EXEC*** is used as a function rather than as a command it returns an error code, for example:

```
DEMON to select employees from database
IF get matching employee
THEN SqlError := EXEC SQL select * from personnel
        where last_name := name END SQL INTO Employees
```

SqlError can have a when-changed method that displays an error message to a user, for example: *"No Records Found."* or *"Connection to the database broken."*.

3.2. Database technology in Kappa-PC

Kappa-PC offers similar database facilities to those provided by Level5 Object [*Kappa-PC User's Guide*, 1992]. Kappa-PC supports reading from and writing to ASCII files. It also provides a set of functions for accessing DBase and Lotus-123 files. Similar to Level5 Object, Kappa-PC provides an interface to an SQL DBMS through a third-party database server (SequeLink from TechGnosis). The final database facility present in Kappa-PC is exchange of data through the Dynamic Data Exchange (DDE) - it is described in more detail in chapter 6.

3.2.1. DBase and Lotus-123 interface in Kappa-PC

Kappa-PC provides a set of functions for accessing DBase and Lotus-123 files. Before any data can be accessed a file must be opened with the built-in **DBOpenFile(*FileName*)** function. For Lotus-123, *FileName* must have a *.wks*, *.wk1*, or *.wrl* extension. For DBase files, *FileName* must have a *.dbf* extension. Many files can be open simultaneously but only one file can be selected. A file opened with **DBOpenFile()** becomes the currently selected file. To select another open file, the **DBSelectFile(*FileName*)** function is used. An open file can be closed with **DBCloseFile(*FileName*)**. For DBase files, Kappa-PC also provides functions for using indexes: **DBOpenIndexFile(*FileName*)**, **DBSelectIndexFile(*FileName*)**, and **DBCloseIndexFile(*FileName*)**.

Both Lotus-123 and DBase files can be accessed using the same functions. The **DBReadCell(*Row#, Col#*)** function returns a value stored in the *Row#* row (record in DBase) and the *Col#* column (field in DBase). The functions **DBGetNumberOfRows()** and **DBGetNumberOfFields()** return the number of rows and columns in a spreadsheet or database, respectively. The **DBReadField(*column*)** function reads data from the column name or column number of the current row, while the **DBGetRowPosition()** function returns the number of the currently active row. The **DBSetRowPosition(*Row#*)** function sets the currently active row. The **DBWriteCell(*Row#, Col#, Value*)** function writes data *Value* to the row *Row#* and to the column *Col#* of a selected file. Both DBase and Lotus-123 files can be searched for rows (records) satisfying a filter expression - **DBFindRecord(*FilterExpr*)**, for example:

```
DBOpenFile("employee");
DBFindRecord("last_name = 'Smith' .AND. first_name = 'Peter'");
```

A more powerful set of functions provided by Kappa-PC supports mapping a row of data from a DBase or Lotus-123 file directly into slots of an object instance in a Kappa-PC application. In order to do this, the following sequence of function calls must be executed to prepare parameters for mapping:

```
SetValue(Global:SlotNames, FirstName, LastName);
SetValue(Global:FieldNames, FIRSTNAME, LASTNAME);
DBSetMapParameters(Global:SlotNames, Global:FieldNames);
```

Now a new instance *Employee1* can be created from the class *Employees* using **MakeInstance(*Employee1, Employees*)**. Finally the function **DBMapRowToInstance()** can be used to supply slot values for this instance from the currently active row in the data file: **DBMapRowToInstance(*Employee1*)**. The opposite mapping of slot values into a row of the data file is also possible. The sequence of function calls is identical to the one used in mapping of a row into slot values except for the last function call. This time the function **DBMapInstanceToRow()** must be used: **DBMapInstanceToRow(*Employee1*)**.

3.2.2. Interface with SQL RDBMS in Kappa-PC

Kappa-PC provides an interface to the SQL relational database using a third-party software package called SequeLink from TechGnosis. SequeLink allows Kappa-PC to connect to a variety of RDBMS, including Ingres, Sybase, Informix, OS/2 EE, Rdb, and DB2 running on a variety of hardware platforms. Kappa-PC's SQL support is based on a client-server architecture where the PC running Kappa-PC is the client and the machine running the SQL DBMS on the network is the server. The use of SQL interface in Kappa-PC is very straight-forward:

1. A database file is opened using **DBOpenFile()**;
2. An SQL **SELECT** command is issued using **DBExecute** function;
3. Various operations can be performed on the table, generated by the SELECT command, using the functions: **DBSetRowPosition()**, **DBGetRowPosition()**, **DBSetMapParameters()**, **DBMapRowToInstance()**, **DBReadCell()**, and **DBWriteCell()**;
4. Steps 2. and 3. can be repeated;
5. The database is closed using the function **DBCloseFile()**.

The Kappa-PC's SQL interface is very powerful and flexible and at the same time easy to use. By relying on a third-party software package, Kappa-PC gains a uniform and reliable SQL interface to a variety of database systems supported by SequeLink.

3.3. KADBASE - a Knowledge-Aided Database System

KADBASE[1], a knowledge-aided database system prototype, is a very flexible interface in which multiple knowledge-based systems and multiple databases can communicate with each other within a distributed engineering computer system [*H.C. Howard & D.R. Rehak*, 1989]. The engineering environment for which the prototype has been developed imposes special

[1] **KADBASE** is not a commercially available product like Level5 Object and Kappa-PC. It is currently a research prototype that has been included in this section to illustrate the next generation of KBS/DBS integration.

requirements on knowledge based systems. The most important requirement is that knowledge based systems must be capable of accessing very large, shared databases. Also the interface must support multiple and heterogeneous knowledge based systems.

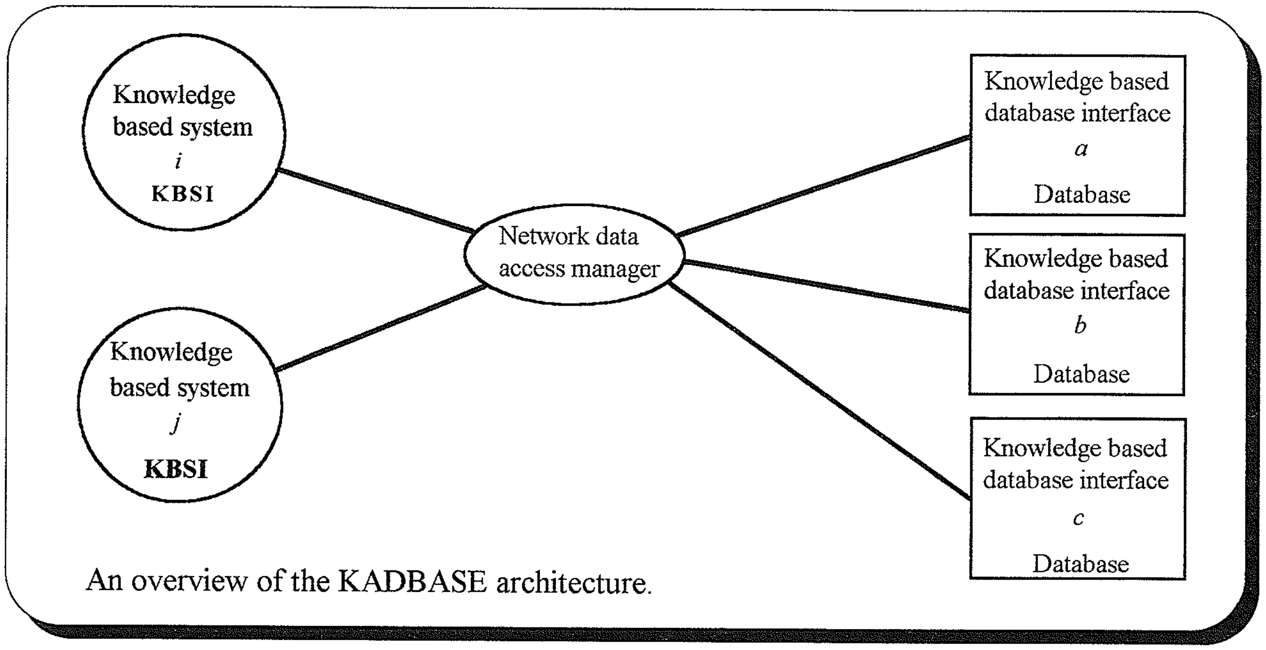

Knowledge based system *i* KBSI
Knowledge based system *j* KBSI
Network data access manager
Knowledge based database interface *a* Database
Knowledge based database interface *b* Database
Knowledge based database interface *c* Database
An overview of the KADBASE architecture.

Figure 3.2

Figure 3.2 shows the following basic components of the KADBASE prototype:

- **The knowledge-based-system interface** (KBSI), part of every knowledge-based system, formulates queries and updates sent to the network data access manager and processes replies from the network data access manager. The KBSI possesses knowledge about the knowledge-based-system context (data space) schema, and uses that knowledge to perform semantic and syntactic translations for queries, updates, and replies.

- **The knowledge-based-database interface** (KBDBI), which acts as an intelligent front end for a basic DBMS, accepts queries and updates from the network data access manager and returns appropriate replies. The KBDBI possesses knowledge about the local database schema and the local language for data manipulation requests. It uses that knowledge to perform semantic and syntactic translations for queries, updates, and replies.

- **The network data access manager** (NDAM), providing the actual interface, receives requests (queries and updates) from knowledge-based systems (through their KBSIs) expressed in terms of the global schema. Using information associated with the global schema, the NDAM locates sources for data referenced in a request and decomposes each request into subqueries or updates to individual target databases. The subrequests are sent to appropriate KBDBIs for processing. Replies from KBDBIs are combined to form a single reply to the original request and sent to the requesting application through its KBSI.

The KADBASE architecture is very powerful and flexible and it is very well suited for large distributed computing environments e.g. CAD and CAM systems. On the other hand it is too large and too complex for an average personal computer and stand-alone workstation systems.KADBASE uses leading-edge technology and it will take some time before it finds its way from the research lab to real-computing environments.

3.4. Conclusions

Level5 Object offers three different methods of integrating with database technology: an interface to DBase (and to Focus databases), an interface to DEC-based Rdb database via SQL, and a DDE-based interface to a third-party database via SQL.

The Level5 Object's implementation of a DBase interface is quite complete and is both easy to understand and use. Its object-oriented approach integrates well with the rest of the system. Most functions needed for efficient database processing (e.g. index files) are supported but there are some severe limitations in the DBase interface. The most serious one is the fact that it requires the use of one particular database system - DBase. Another limitation is that only 10 files can be open simultaneously and when index files are used only six files can be open. Also no relations between files can be established so more programming is required for related files which makes the knowledge base less legible. FIND statements cannot be nested which makes searching two or more related files even more cumbersome. Record locking is also not supported. In summary, the DBase interface in Level5 Object is sufficient for developing non-portable (for IBM PC and DBase only) expert systems with limited (not complex) database processing.

A more flexible database interface offered by Level5 Object is its SQL-based link to a VAX-based Rdb database. The use of a third-party, well-established, database system makes the interface very reliable and complete. The use of SQL further increases the interface flexibility and portability. On the other hand, the interface works only with a VAX-based Rdb database - a major limitation. Interface to other third-party database systems using this method would be desirable.

The most flexible database interface supported by Level5 Object is through the Dynamic Data Exchange (DDE). It also uses SQL and a third-party database system but it is more generic than the previous method in that it can work with any database system supporting both SQL and DDE. On the other hand it is limited to an IBM PC hardware platform and Microsoft Windows environment. This method is described in detail in chapter 6.

Kappa-PC offers a very similar set of database integration facilities to those of Level5 Object. It directly supports both DBase III and Lotus 1-2-3 file formats, including DBase indexes, and provides a set of built-in functions that give access to them. Similar to Level5 Object, Kappa-PC offers an SQL interface to a variety of database systems through a third-party software package - SequeLink. SequeLink provides access to many popular database systems as opposed to the Level5 Object's SQL interface that supports only VAX-based Rdb databases. Kappa-PC also provides a DDE protocol which is more complete than that of Level5 Object (both DDE client and DDE server are implemented).

Many commercial expert system tools available today provide a rich set of database integration facilities but none is complete and perfect for all applications. A user should consider requirements of the actual application when choosing the right tool. Expert system tools are evolving rapidly in recent years and we can expect new methods of integrating knowledge-based systems with database technology. Client-server architectures are becoming more and more popular as well as powerful and we can expect the future expert system tools to evolve in this direction. An example of this is the CADBASE prototype discussed in this chapter that uses a very complex client-server architecture in a network environment. With growing popularity and advances in object-oriented technologies, including object-oriented databases, the next generation of expert system tools will develop object-oriented interfaces to databases that will hide the implementation details and support easier extensibility.

4. Calling External Programs/Functions from a Knowledge-Based System - Implementation

Conventional programming technology can be used for solving well-defined problems for which precise algorithms can be found. Development of a conventional application typically is based on a classic design-implement-test cycle. Every aspect of the application must be defined in terms of a precise algorithm. Conventional programs are very fast, they can be optimized for speed, and give accurate results. Conventional programs are typically implemented in procedural, general-purpose, languages like C, C++, Pascal, Fortran, Basic, PL/1, or Assembler. In contrast to conventional programming technology, knowledge-based technology can deal with uncertain and complex problems. Development of a knowledge-based system does not start with a complete design, instead incremental development and rapid prototyping techniques are used. As opposed to a conventional system, knowledge is explicitly separated from the problem solving part of a knowledge-based system. This enables easier modification and maintenance of knowledge. However, these benefits of knowledge-based technology are achieved at a price of decreased efficiency. Also, knowledge-based systems do not have a rich set of math functions that conventional languages have.

The ability to call an external program or function from within a knowledge-based system can extend the system's capabilities and can reduce development time and costs significantly. It is useful for complex math calculations, for a complex graphics display, for printing a report, etc. Two levels of implementation are possible:

- calling an external program/function without further communication;
- calling an external program/function with the parameters and results passing.

The first method is very easy to use and does not require any special support from a caller other than the ability to start an external program. It can be used, for example, to perform some background tasks like printing. The second method is much more flexible and useful but also much more difficult to implement, especially when the parameters and results are passed through memory as opposed to disk files. Typically it requires some support on the part of a caller in order to pass parameters and receive results.

The following discussion of external program/function calling methods will pertain to MS Windows and MS DOS environments. The most flexible and tool-independent method of calling external functions in Microsoft Windows environment is through the use of Dynamic Link Libraries (DLLs). In Microsoft Windows, a DLL is a library of object code in the form of functions available for external access, together with a function access table. The functions within a library can be dynamically accessed from another program at runtime without having previously linked the library with the program. This feature allows high level programs (e.g. expert system shells), which usually do not need compilation, to access functions supplied by DLLs. Another nice feature of DLLs is that they are independent of the programming language - they can be written in C, C++, Pascal, Basic, or any other language that supports DLLs.

4.1. Calling external programs/functions in Level5 Object

Level5 Object provides two commands for calling external programs [*Level5 Object User's Guide*, 1990]:

- **ACTIVATE**: used when an external program is to be called only once;
- **ESTABLISH:** used when an external program will be called more than once (it will stay in memory after the first call).

Two types of programs can be called from Level5 Object: EXTERN and SERVER. EXTERN program is an external program that is called with optional command-line parameters and no further communication exists between the external program and Level5 Object. It can be any MS DOS or MS Windows application. SERVER is an external program that can receive parameters from Level5 Object and can pass back the results upon termination (parameters and results are passed through memory). SERVER programs must be written in Microsoft C language and must conform to special rules imposed by Level5 Object.

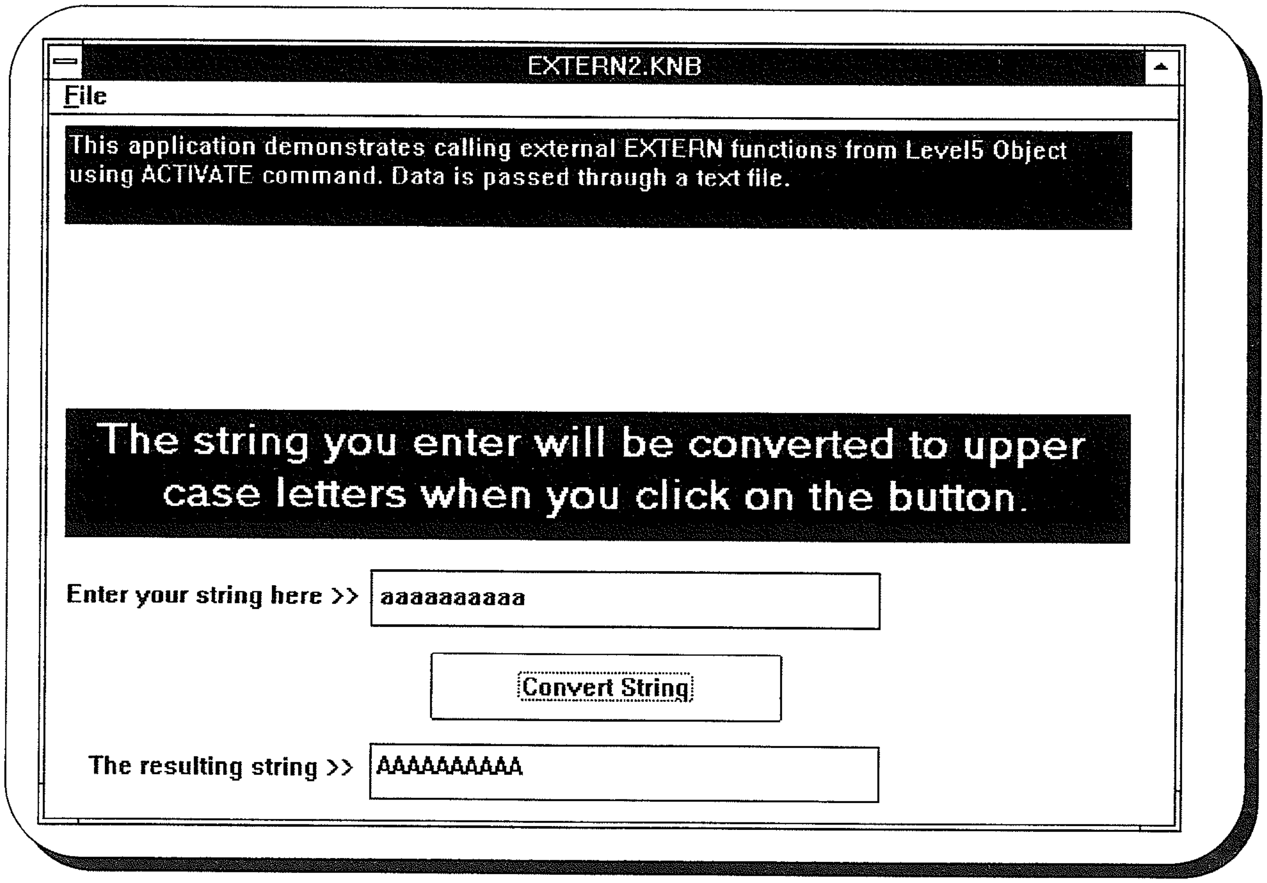
4.1.1. Calling an EXTERN program

An EXTERN program does not have to be written specifically to communicate with Level5 Object. It can be any MS DOS or MS Windows application. Command line parameters can be sent to the program but no further communication exists. The following is an example of a call to an EXTERN program:

```
ACTIVATE "IPU, EXTERN, NOTEPAD.EXE"
COMMAND filename[1]
```
where filename is a STRING attribute containing the name of a text file.

[1] This PRL code can be used in a rule, in a demon, or in a when-changed or when-needed method.

This command, when executed by the Level5 Object's inference engine, will start the MS Windows program Notepad and pass the filename parameter as a command-line argument. Notepad, a simple text editor, will open a file, the name of which was passed on the command-line, for editing. Level5 Object will continue its processing but the user can switch to the running Notepad to edit the file and after that switch back to Level5 Object.



This application demonstrates calling external EXTERN functions from Level5 Object using ACTIVATE command. Data is passed through a text file.

The string you enter will be converted to upper case letters when you click on the button.

Enter your string here >> aaaaaaaaaa

The resulting string >> AAAAAAAAAA

Figure 4.1

EXTERN2.KNB application (figure 4.1) demonstrates another example of how to call an external function using this method. After a user enters a string and clicks on the Convert String button, Level5 Object calls an external program passing the string to it through a text file. The external program reads the string from the file, converts it to upper case, and passes the converted string back to Level5 Object through the same file. The string is then displayed in the main window.

Level5 Object accomplishes the above using the following code:

```
WHEN CHANGED
 BEGIN
  action OF file 1 IS open new := TRUE
  write line OF file 1 := str1
  action OF file 1 IS close := TRUE
  ACTIVATE  "IPU,EXTERN,C:\PRG\L5O\SERVER\TOUPPER2.EXE"
  action OF file 1 IS open old := TRUE
  read line OF file 1 := TRUE
  str2 := current line OF file 1
  action OF file 1 IS close := TRUE
 END
```

Level5 Object creates an ASCII file by setting the *action OF file 1* attribute to *open new*. Then it writes the *str1* string to the file and closes the file by setting the *action OF file 1* attribute to *close*. Next, Level5 Object calls the external program using *ACTIVATE* statement (the external program becomes the active task in the MS Windows environment). After the external program terminates (or yields control to MS Windows) MS Windows returns control back to Level5 Object. Level5 Object then opens the file by setting the *action OF file 1* attribute to *open old*, reads the line from the file to the result string *str2*, and closes the file.

The external program used in this example is an MS Windows application written in Microsoft C but it could be any Windows or DOS program not necessarily written in C. The code is shown below:

```
/* FILE : TOUPPER2.C
  DESCRIPTION    : Converts all characters from a string to upper case.
                   External program called by LEVEL5 OBJECT EXTERN2.KNB.
                          String is passed through the EXTERN2.DAT text file.
*/
#define fName "extern2.dat"
#include <windows.h>
#include <ctype.h>
#include <stdio.h>

int  PASCAL    WinMain(HANDLE,HANDLE,LPSTR,int);
void read_string(char *s)
{  FILE *inp;
   if ((inp = fopen(fName,"rt")) != NULL)
      fgets(s,255,inp);
   fclose(inp);
}

void write_string(char *s)
{  FILE *out;
   if ((out = fopen(fName,"wt")) != NULL)
      fputs(s,out);
   fclose(out);
}

int PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine, cmdShow)
HANDLE hInstance, hPrevInstance; LPSTR  lpszCmdLine; int    cmdShow;
{  char    sInput[255];
   char    sOutput[255];
   int     status;
   char    *psInput,*psOutput;

  /* read a string from a predefined file*/
  read_string(sInput);
  psInput = sInput;
  psOutput = sOutput;
  while (*psInput)
  {
    *psOutput = *psInput;
    *psOutput = toupper(*psOutput);
    psInput++;
    psOutput++;
  }
  *psOutput = '\0';

  /* return a string to the caller*/
  write_string(sOutput);
  return TRUE;
}
```

This program will work with any other program not just a Level5 Object application.

4.1.2. Calling a SERVER program

A SERVER program is written specifically to communicate with Level5 Object. It calls functions that read the values of attributes in SEND statements (parameters passed from Level5 Object) and that write values to RECEIVE statements (results passed back to Level5 Object). A SERVER program must be an MS Windows program. The following is an example of a call to a SERVER program from Level5 Object:

```
ACTIVATE "IPU, SERVER, C:\LEVEL5\PROG.EXE"[1]
SEND name OF Employee
SEND employee.sex IS male
RECEIVE salary OF employee
```

The ACTIVATE command starts the server program PROG.EXE and establishes communication between the server and Level5 Object. Next, Level5 Object sends two parameters to the server using SEND commands which should have the corresponding *l5_read* statements. After the last SEND command Level5 Object yields control to the server. After finishing the processing, the server returns the result to Level5 Object using one of *l5_write* statements. Level5 Object reads the result using the RECEIVE command. At this point the server terminates and the control is returned to Level5 Object. If a Level5 Object application does not use the RECEIVE command (it does not expect a result from a server) the server should use *l5_quit* statement to inform Level5 Object that it can continue processing.

[1] This PRL code can be used in a rule, in a demon, or in a when-changed or when-needed method.

Writing a SERVER program involves following the rules described below. A library of functions, called L5SERVER.LIB, is supplied with Level5 Object to allow a SERVER program to communicate with Level5 Object. A header file, L5SERVER.H, contains the function prototypes that should be included in the program. Level5 Object requires that the SERVER program be written in Microsoft C 5.0 or higher. Also, the MS Windows SDK is needed. In writing a SERVER program the following steps should be followed:

- Establishing a communication path with Level5 Object:
  - **l5_access(programName)** function should be called at the beginning of the server program where **programName** is the name of the server e.g. PROG.EXE;
- Reading attribute values sent by Level5 Object:
  - read calls must match the SEND statements of a Level5 Object application (**cf** means a Certainty Factor);
  - **l5_read_num(cf,num)** function reads a NUMERIC attribute;
  - **l5_read_logical(cf)** function reads a SIMPLE, COMPOUND, and MULTICOMPOUND attributes;
  - **l5_read_string(cf,string,length)** function reads a STRING attribute;
- Writing attribute values to Level5 Object (passing back results):
  - write calls must match RECEIVE statements of a Level5 Object application (**cf** means a Certainty Factor);
  - **l5_write_num(cf,num)** function passes a NUMERIC value to Level5 Object;
  - **l5_write_logical(cf)** function passes a SIMPLE, COMPOUND, and MULTICOMPOUND attributes to Level5 Object;
  - **l5_write_string(cf,string,length)** function passes a STRING attribute to Level5 Object;
- Synchronizing with Level5 Object:
  - it is only needed when no SEND and RECEIVE statements are used in an ACTIVATE (or ESTABLISH) command;
  - **l5_quit()** function should be used to inform Level5 Object that it can continue processing.

The EXTERN.KNB application demonstrates how to use a SERVER program in Level5 Object. It accomplishes the same task as EXTERN2.KNB - converting a string to upper case, but instead of passing the string through a file it passes it through memory using the ACTIVATE command. The following code is used:

```
WHEN CHANGED
 BEGIN
  ACTIVATE  "IPU,SERVER,C:\PRG\L5O\SERVER\TOUPPER.EXE"
   SEND  str1
   RECEIVE  str2
 END
```

The TOUPPER.C program is written in Microsoft C - no other compiler can be used. It must also be linked with L5SERVER.LIB library. The string is passed through memory which is much more efficient than passing it through a file. The code is shown below:

```
/* FILE  : TOUPPER.C
  DESCRIPTION     : Convert all characters from a string to upper case.
                    External program called by LEVEL5 OBJECT's EXTERN.KNB.
*/

#include <windows.h>
#include <ctype.h>
#include "l5server.h"

int  PASCAL   WinMain(HANDLE,HANDLE,LPSTR,int);

int PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine, cmdShow)
HANDLE hInstance, hPrevInstance; LPSTR  lpszCmdLine; int    cmdShow;
{  char    sInput[L5SERVER_MAX_STRING_LENGTH];
   char    sOutput[L5SERVER_MAX_STRING_LENGTH];
    int    status;
    char   *psInput,*psOutput;
    char    cf;

 /* establish the communication link with LEVEL5 OBJECT */
 if ((status = l5_access("TOUPPER.EXE")) < 0) 1
    return FALSE;

 /* read a string from a LEVEL5 OBJECT's SEND statement */
 l5_read_string(&cf, sInput, L5SERVER_MAX_STRING_LENGTH);

 psInput = sInput;
 psOutput = sOutput;
 while (*psInput)
 {
     *psOutput = *psInput;
     *psOutput = toupper(*psOutput);
     psInput++;
```

1 **l5_access**, **l5_read_string**, and **l5_write_string** are supplied in the L5SERVER.LIB library.

```
     psOutput++;
   }
   *psOutput = '\0';

   /* return a string to the LEVEL5 OBJECT's RECEIVE statement */
   l5_write_string(cf, sOutput);
   return TRUE;
 }
```

The EXTERN1.KNB application is almost identical to EXTERN.KNB but instead of using Level5 Object's ACTIVATE command it uses the ESTABLISH command. This is a more efficient method when a server program is called more than once. The first call will load the server program into memory and it will stay there, so all subsequent calls will not require reloading it. This method requires that a server be written in a specific way - it must stay in a loop until it cannot communicate with Level5 Object. This is shown below:

```
/* FILE  : TOUPPER1.C
   DESCRIPTION    : Convert all characters from a string to upper case.
                    External program called by LEVEL5 OBJECT's EXTERN1.KNB.
*/

#include <windows.h>
#include <ctype.h>
#include "l5server.h"

int  PASCAL  WinMain(HANDLE,HANDLE,LPSTR,int);

int PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine, cmdShow)
HANDLE hInstance, hPrevInstance; LPSTR  lpszCmdLine; int    cmdShow;
{  char   sInput[L5SERVER_MAX_STRING_LENGTH];
   char   sOutput[L5SERVER_MAX_STRING_LENGTH];
   int    status;
   char   *psInput,*psOutput;
   char   cf;

 /* establish the communication link with LEVEL5 OBJECT */
 if ((status = l5_access("TOUPPER1.EXE")) < 0)
   return FALSE;
/* Continue the loop until l5_read_string() or l5_write_string fails() */
while (status >= 0)
{
    /* read a string from a LEVEL5 OBJECT's SEND statement */
    status = l5_read_string(&cf, sInput, L5SERVER_MAX_STRING_LENGTH);

   psInput = sInput;
   psOutput = sOutput;
   while (*psInput)
   {
      *psOutput = *psInput;
      *psOutput = toupper(*psOutput);
      psInput++;
      psOutput++;
   }
   *psOutput = '\0';

   /* return a string to the LEVEL5 OBJECT's RECEIVE statement */
   if (status >= 0)
     status = l5_write_string(cf, sOutput);
}
return TRUE;
}
```

4.2. Calling external programs/functions in Kappa-PC

Kappa-PC provides several methods of calling external functions. The simplest method allows executing an external MS DOS or MS Windows program from within a Kappa-PC application. This is accomplished by the **Execute()** function, for example:

```
Execute("c:\windows\write.exe", "myfile.txt");
```

This function call will start the MS Windows Write program which will open *mytext.txt* file for editing. Up to three arguments can be used with **Execute()** function.

The second method is Kappa-PC's ability to generate a C code for the Kappa-PC-based application using the KAL[1] compiler. Once the C code is generated it can be extended by any C functions that are needed. Next, the C code must be compiled and linked into an MS Windows DLL using a standard C compiler and linker able to produce DLLs. Once the DLL is created it can be run by Kappa-PC, for example: *win kappa myapp.dll*. Kappa-PC is still needed to run a compiled application but it will work much faster than a KAL-based application. The whole procedure of generating a C code from a KAL code is not simple. First, a *kalmake.cfg* file must be created (its contents depends on the compiler and linker used). Next, the KAL compiler can be used to generate the C code. This can result in a number of error messages because some KAL functions cannot be compiled. The user must modify the KAL code (by substituting those forbidden functions with others) until no errors are signaled. Once the KAL code is converted to the C code, the user-defined C functions can be added (certain guide lines must be followed). Finally, the C code can be compiled and linked into a DLL.

The next two methods of calling external functions are very similar to each other and both involve using DLLs. Kappa-PC provides a set of KAL (Kappa Application Language) functions to directly access functions from Dynamic Link Libraries (DLLs). The user only has to know the name of a function, the types of parameters passed to the function, and the type of a return value. In order to make a DLL function available to the KAL application the function has to be registered by calling the **DeclareDLL** function: **DeclareDLL(KalName, DLLName,**

[1] **KAL** stands for Kappa-PC Application Language.

LibName, ReturnType, ArgType,...). Once the DLL function has been registered it can be used as if it were a built-in KAL function. Other DLL related functions provided by Kappa-PC are:

- **DeleteDLL(KalName)**: deletes a previously registered function;
- **SetPointerData(pointer, offset, type, value)**: sets the memory value at the address of ***pointer+offset*** to ***value***;
- **GetPointerData(pointer, offset, type)**: gets the data value at the memory address of ***pointer+offset***;
- **SizeOf(type)**: returns the size of the specified data type in bytes.

By using the above functions it is possible to call functions from both off-the-shelf DLLs (e.g. those that are part of Microsoft Windows) and user-written DLLs. The following example shows how to register and use several DLL functions from the Microsoft Windows' Kernel DLL. It also demonstrates how to use the **GetPointerData** function in order to retrieve the result of a previously called DLL function [*Kappa-PC Advanced Topics*, 1992].

Example:

```
DeclareDLL(Alloc, GlobalAlloc, kernel, HANDLE, INT, DWORD);
DeclareDLL(Lock, GlobalLock, kernel, POINTER, HANDLE);
DeclareDLL(Unlock, GlobalUnlock, kernel, INT, HANDLE);
DeclareDLL(Free, GlobalFree, kernel, HANDLE, HANDLE);
DeclareDLL(GetWinDir, GetWindowsDirectory, kernel, INT, POINTER, INT);

Global:Handle = Alloc(2, 256);
Global:Pointer = Lock(Global:Handle);
GetWinDir(Global:Pointer, 256);
GetPointerData(Global:Pointer, 0, STRING);
Unlock(Global:Handle);
Free(Global:Pointer);
DeleteDLL(Alloc);
DeleteDLL(Lock);
DeleteDLL(Unlock);
DeleteDLL(Free);
DeleteDLL(GetWinDir);
```

The first five calls to the **DeclareDLL()** function register with Kappa-PC five functions from MS Windows kernel DLL. These functions are: **GlobalAlloc(), GlobalLock(), GlobalUnlock**(), **GlobalFree(),** and **GetWindowsDirectory()**. After registration these functions are used as if they were built-in KAL functions. At the end all functions are deleted using **DeleteDLL()** function calls - they are no longer available to the KAL application.

Another method of using DLL-based functions in a KAL application is very similar to the previous method. The only difference is that instead of using the *DeclareDLL()* function to register external functions with Kappa-PC the external functions are automatically registered when Kappa-PC starts up. This makes the external functions become the part of KAL just as built-in functions are. In order for Kappa-PC to automatically register external DLL functions a special procedure must be followed. First, the new functions must be registered in the *udllinit.c* file provided by Kappa-PC. The *template.def* file, also provided with Kappa-PC, must then be customized - it should declare all new functions and export them. Next, the DLL containing new functions should be created (using the provided *template.mak* make file). Finally, the *kappa.ini* file must be edited - it should register all new functions with Kappa-PC kernel.

4.3. Conclusions

All the functionality necessary to call external programs is provided by Level5 Object. Calling EXTERN program is much easier and does not impose any special requirements on the external program. Parameters can be passed on the command-line, through ASCII files, or through DBase files. If passing parameters through the files is not efficient enough then SERVER program must be used. Writing a SERVER program imposes special requirements on the program and can be cumbersome but it allows passing parameters and results efficiently through memory. Also a SERVER program must be written in Microsoft C. Level5 Object should support other most popular C compilers such as Borland C++.

Kappa-PC provides a more generic mechanism of calling external functions than Level5 Object. The use of DLLs is more convenient than the use of conventional libraries because no linking is necessary for the application to use DLL-defined functions. DLLs also can be created in any programming language that supports DLLs (e.g. C, C++, Pascal). The interface to DLLs consists of a few easy-to-use functions built into Kappa-PC. The only step required, before an external DLL function can be called, is registering the function. All expert system tools running in the Microsoft Windows environment should support DLLs.

The ability to call external code, written in a conventional programming language, greatly enhances the power and flexibility of a knowledge-based system. It allows a knowledge-based application to efficiently perform tasks, such as complex numerical functions or graphics display, better suited to conventional-computing technology. Methods discussed in this chapters illustrate how a knowledge-based application can call external code. Next we will look at how an external (conventional) code can call knowledge-based tools.

5. Embedded Solutions - Implementation

Embedding a knowledge-based application in a conventional system can extend the capabilities of an existing conventional application in a way that is transparent to a user. For example, an existing database system can be extended by a knowledge-based front-end. As a result the system will become easier to use but the user will not even notice the existance of the knowledge-based component. This method of integration can be used as a transition vehicle from an entirely conventional application to a more knowledge-based oriented application without causing a shock to end-users.

Embedding an expert system in a conventional application means that the conventional application has access to the expert systems knowledge base functions and working memory. The conventional program must be able to do all of the following:

- insert initial facts into the expert system's working memory;
- call the inference engine to start reasoning;
- select a control strategy (forward chaining, backward chaining);
- access and/or modify the goals on the agenda;
- display all the output messages, prompts, and explanations generated by the expert system;
- pass the user's input to the expert system (e.g. responses to the prompts);
- interrupt the reasoning process of the expert system;
- access the results of the reasoning;
- examine the state of the reasoning process of the expert system.

There are two basic methods of embedding an expert system in a conventional application, a tight embedding where the expert system's object code is linked into the conventional application, and a loose embedding where a conventional application communicates with an expert system either using some form of application interface (e.g. using Microsoft Windows DLLs) or using some form of client-server architecture where the expert system component acts as a server (the client-server architecture approach is described in the next chapter).

5.1. Embedding in CLIPS

Tight embedding of an expert system in a conventional program (e.g. written in the C language) allows for the closest and most efficient, in terms of program execution, integration of both technologies. The implementation of this approach is typically quite difficult. Both systems must be linked together to form one executable module. Knowledge of the internal structures of both systems and code modifications are necessary. The major benefit is a very tight integration. The expert system can call functions directly from the conventional system, and the conventional system can call the expert system's inference engine directly and can access the working memory of the expert system component directly. Also a programmer, having access to the expert system's source code, can modify the expert system's internal functions to customize its operation (this can be very difficult and tricky even for experienced programmers). This approach to embedding is offered by CLIPS [*William Mettrey*,February 1991].

In order to create an embedded CLIPS application it is necessary to compile the knowledge-base to C source code using the *rules-to-c* option supplied by CLIPS. Next, the C source code generated from the knowledge-base can be recompiled and linked with CLIPS run-time libraries as well as with other conventional C code. The result is an executable code.

Also, CLIPS comes with the C source code that can be customized by the developer and recompiled resulting in a customized version of CLIPS.

5.2. Embedding in Kappa-PC

A different method of embedding is used in Kappa-PC. The Kappa-PC kernel, containing all the essential elements of the tool, is structured into several DLLs:

- the ***foundation layer DLLs*** provide memory allocation and management, error and exception management and recovery, list management, and a string table, together with some general utilities;

- the ***object layer DLLs*** provide for the management of objects, slots, methods, functions, rules, and goals; it contains all the inheritance management as well as Kappa-PC Application Language interpretation modules;

- the ***KAL library layer DLL*** provides a set of functions that are registered to be accessed from the KAL language;

- the ***rule system layer DLLs*** contain the rule network management, as well as the forward and backward engines;

- the ***active images management layer DLL*** contains the functionality to manage session windows, images, and menus;

- the ***database interface layer DLLs*** provides interface to various database systems, such as Lotus 1-2-3, dBase III, and through SQL to relational databases using TechGnosis product;

- the ***tools layer DLLs*** contain functions for managing development tools, such as the KAL interpreter window and the KALView Debugger.

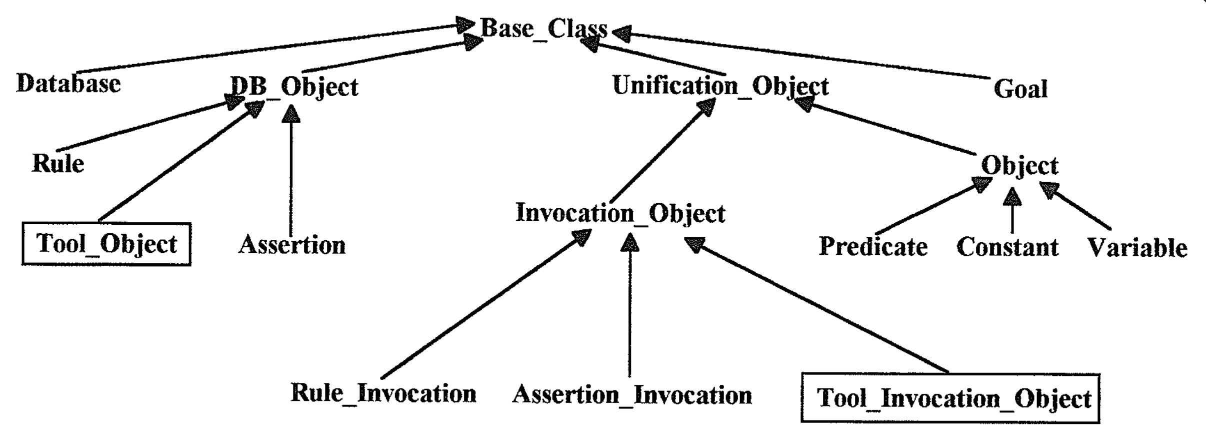
This structured DLL system allows for relatively easy access to the Kappa-PC functionality from other Microsoft Windows-based conventional applications. The only requirement imposed on a conventional program is that it must follow the Microsoft Windows rules of using DLLs.

Kappa-PC currently can support only one application at a time - no more than one application can access Kappa-PC's kernel DLLs [*Kappa-PC Advanced Topics*, 1992]. In order to link the Kappa-PC kernel libraries a developer must use the appropriate "include files" (supplied by Kappa-PC) and link the application with the Kappa-PC-supplied libraries (LIB files). An alternative method is to load the Kappa-PC libraries dynamically, using the MS Windows LoadLibrary() function, and then call Kappa-PC kernel functions using the GetProcAddress() function.

Another advanced feature of Kappa-PC is its ability to compile KAL applications into C code which in turn maybe integrated with other C code, compiled and linked into a robust executable code (in the form of a DLL). The process of generating C code from a KAL-based application is described in detail in chapter 4. Despite Kappa-PC's ability to create classes and instances dynamically e.g. **MakeClass()**, **MakeInstance()**, pointers to objects are not supported which constitutes a major limitation.

5.3. Embedding using Object-Oriented Programming Technology

The CAD Inference Engine (CADIE) implements a rule-based inference engine designed to be easily embedded in other applications, specifically CAD tools developed in C++[*David W. Franke*, Dec. 1990]. CADIE uses features of object-oriented programming to achieve integration. CADIE's unique feature lies in its integration of application data structures with rules and assertions. This is accomplished by using an object-oriented system for both its knowledge representation (rule-based) and inference engine. This is shown by the following class hierarchy (figure 5.1):



The CADIE class hierarchy.

Figure 5.1

This hierarchical organization of CADIE (figure 5.1), which relies on such features of object-oriented programming like inheritance and polymorphism (the ability of different objects to respond to the same interface), enables an easy integration of CADIE into other applications written in C++. Two special classes Tool_Object and Tool_Invocation_Object are provided as an interface to an application. These classes define member functions required by the inference engine of CADIE, providing transparent access to application-defined objects. An application developer has to derive application-specific objects (classes), that will be accessed by the inference engine, from the class Tool_Object. He also has to create, for every application-specific object derived from the Tool_Object class, a new class derived from the class Tool_Invocation_Object. This new class has to redefine three member functions:

```
virtual char  **cadie_One_Place_Predicate_Names();
virtual char  **cadie_Two_Place_Predicate_Names();
virtual int  cadie_Unify_Two_Place_Predicate(cadie_Object *, cadie_Substitution *, int);
```

The approach to embedding expert system capabilities in conventional systems used in CADIE enables tight, easy-to-use and efficient integration of both technologies. CADIE itself is not as powerful and flexible as other well-established expert system tools so it could not be used for large-scale expert system development. On the other hand, CADIE demonstrates the potential benefits of object-oriented programming for expert system tools. If any of the leading expert system tools currently on the market could be ported to C++ it could use the approach of CADIE to embedding. Other interfaces (e.g. database interface) of such a tool could benefit in a similar way from object-oriented programming.

5.4. Conclusions

The approach to embedding used in Kappa-PC seems to be superior to the one used in Clips (tight embedding). It is far easier to use since it does not require understanding the source code of an expert system tool. The efficiency of functions called through DLLs is almost equal to the functions linked directly into the application. The DLL approach to embedding should satisfy the needs of even the most sophisticated applications providing the functions accessible through DLLs cover all aspects of the expert system tool. The most elegant and the easiest-to-use method of embedding is used in CADIE. Its power comes from object-oriented programming, specifically from inheritance and polymorphism. Future expert system tools should be ported to object-oriented languages and fully utilize the object-oriented programming mechanisms.

One very important feature of object-oriented programming is the ability to create objects dynamically and to access those objects by pointers. Pointers to objects can be stored in simple variables or in more complex structures like arrays of pointers or linked lists. Some objects could be related to other objects by having a pointer to an object as an attribute. Such features would significantly increase an expert system's flexibility and efficiency. Neither Level5 Object nor Kappa-PC supports pointers to objects.

6. Client-Server Architecture - Implementation

A Client-Server architecture is such a computing environment in which two or more applications communicate with each other using a common communication protocol. This requires a multitasking environment (e.g. MS Windows). The communication protocol must be designed in such a way that in any conversation one application assumes a role of a client (or master) and another application assumes a role of a client (or a slave). Typically a client application is responsible for initialization and termination of a conversation. MS Windows, which provides a multitasking environment, has a built-in communication protocol called DDE (Dynamic Data Exchange) that can be used as a basis for the Client-Server architecture.

A Client-Server architecture is the most flexible method of integrating expert system technology with conventional programming technology. On the other hand this method imposes the highest requirements on the operating system of all the methods discussed previously:

- the operating system must provide multitasking capability in order to allow at least two applications to execute simultaneously (a client and a server);

- the operating system must support inter-process communication to allow exchanging data and commands between a client and a server;

Microsoft Windows fulfills the above requirements. It is a multitasking system and provides a Dynamic Data Exchange (DDE) protocol which allows two or more applications to exchange both data and commands through memory. Further discussion of a client-server techniques will be based on Microsoft Windows and the DDE protocol.

All applications used in the DDE conversation must have been written according to Microsoft specifications for the DDE protocol. Each DDE conversation involves two applications. The application which starts the conversation assumes a role of a client and the responding application becomes a server. The client initiating the conversation must specify an **application** that will become a server, a **topic** (e.g. a file name), and an **item** (e.g. a column in a spreadsheet). The client must also specify one of three types of links for the selected topic and item:

cold link:	the server sends data only on the client's request;
warm link:	the server informs the client about any change in data and then the client can request the server to send it;
hot link:	the server sends data to the client every time when data changes.

The cold link is useful when the client will need data from the server only at some predictable points in time and when the client should not be interrupted (even when new data is available) because it performs some important tasks. It can also be used when data in the server application does not change very often and when the client does not need to react quickly to the changes in data. The warm link is useful when the client wants to be informed about data changes immediately but it does not want to be interrupted when performing important functions. The client can request data from the server when it is convenient for it. The hot link is useful when the client wants to receive data immediately after the data changes take place.

The client can also send commands to the server requesting the server to perform some actions. Finally the client is responsible for terminating the conversation. One application can be involved in several DDE conversations simultaneously and assume both client and server roles.

6.1. Client-Server Architecture in Level5 Object

Level5 Object supports the DDE protocol through the **DDE** system class which has the following structure:

```
CLASS DDE
      WITH app STRING
      WITH topic STRING
      WITH item STRING
      WITH active SIMPLE[1]
      WITH attachment REF[2]
      WITH data ready SIMPLE
      WITH action COMPOUND[3]
              poke,
              request,
              execute
      WITH link COMPOUND
              hot,
              warm,
              cold
      WITH append SIMPLE
      WITH autostart SIMPLE
      WITH time out INTERVAL[4]
      WITH show error SIMPLE
      WITH default error handling SIMPLE
      WITH error NUMERIC
      WITH error message STRING
```

Level5 Object 2.5 only supports half of the protocol with Level5 Object being a client. In Level5 Object, a DDE conversation with another application is represented as an instance of the

[1] **SIMPLE** attribute type corresponds to Boolean in Pascal.
[2] **REF** attribute type is a pointer to an attribute or to a class instance.
[3] **COMPOUND** attribute type is like Enumerated type in Pascal, for example the action attribute can only assume one of the following values: poke, request, or execute.
[4] **INTERVAL** attribute type represents a duration of time in days, hours, minutes, seconds, and miliseconds.

DDE system class. This class insulates the user from the complexities of DDE protocols and reduces the chance of errors when establishing a DDE conversation. The attributes of the DDE class that are required to initiate a conversation are **app**, which is the server application name; **topic**, which is an agreed upon topic of a conversation; and **item**, which can be any number of topic-specific items. For example, a request to Microsoft Excel for a range of spreadsheet cell values would have the following conversation attributes:

```
app = "Excel";
topic = "Stock.xls";
item = "R1C1:R5C20" (range of cells).
```

When a knowledge base is run, Level5 Object examines each instance of the DDE system class. If the **app** and **topic** attributes have values, and the **active** attribute has a value of TRUE, then Level5 Object attempts to establish a conversation. Further, if the **item** and **action** attributes have values, then data is exchanged. At any time during a session, **active OF DDE** may be set to TRUE to initiate a conversation. Setting **active** to FALSE terminates a conversation. The attributes of instances of the **DDE** class can be given values either at edit time (after the knowledge base has been created) or at run-time through rules, demons, and methods.

Level5 Object supports all types of DDE links: hot, warm, and cold. In a cold link, a client receives data from a server only upon request by the client. In a hot link, whenever data changes in the server, the server automatically sends the new values to the client. In a warm link, the client notifies the server whenever the specified data item changes. In Level5 Object, a warm link is set by initiating a conversation, entering a value for the item attribute that is to be monitored, and then setting the compound item **link OF DDE IS warm** to TRUE. Whenever the data item in the server changes, the server informs Level5 Object by setting the attribute **data ready** to TRUE. The **data ready** attribute can then be used as a trigger for a demon in order to obtain the data, e.g.:

```
item OF DDE_to_Excel := "R1C1"
link OF DDE_to_Excel IS warm
. . .
DEMON get data
IF data ready OF DDE_to_Excel
THEN action OF DDE_to_Excel IS request
```

In this example, one instance of the DDE system class, DDE_to_Excel, is used. First, **link OF DDE_to_Excel** sets the warm link, then whenever the data in R1C1 of the Excel spreadsheet changes, the **data ready** attribute for that instance is set to TRUE. When this happens, the demon[1] fires, and the spreadsheet values are requested from the server.

The data that Level5 Object receives and transmits through the DDE protocol is instance data[2]. The **attachment OF DDE** attribute determines what data is sent or received. There can be four attachment types:

1) The attachment can be an attribute of an instance.
When the attachment is an attribute of a specific instance, a single value is being exchanged, such as one cell in a spreadsheet, one block of text in a document, etc.

[1] Demons in Level5 Object are rules that are applied using a data-directed (forward chaining) algorithm.
[2] Level5 Objects sends and receives values of class instances.

2) The attachment can be an attribute of a class.
When the attachment is an attribute of a class, Level5 Object makes a new instance of the class for each value received. In a spreadsheet, these new instances create a column of data consisting of all of the instance values of that attribute.

3) The attachment can be an instance.
When the attachment is an instance, each attribute of this instance will exchange its value. It is analogous to a row in a table. No new instances are created. The data received is matched to the subsequent attributes in the instance.

4) The attachment can be a class.
When the attachment is a class, a whole table of data is exchanged. The instances of the class represent rows, and the attributes represent columns. Instances are created for each row received.

6.2. DDE Examples in Level5 Object

Two Level5 Object example applications will be described in order to illustrate some of the capabilities of Dynamic Data Exchange. The first example application, CONFIG.KNB, comes with Level5 Object. The second application, DDE.KNB, has been created for the purposes of this thesis.

6.2.1. CONFIG.KNB Application

CONFIG.KNB is an example application that comes with Level5 Object (figure 6.1). On startup it launches, using the **DDE autostart** attribute, two other programs: Microsoft Excel - a spreadsheet, and Q + E - a relational database. Using two separate DDE system class instances, Level5 Object asks Microsoft Excel to open the CONFIG.XLS spreadsheet and the CONFIG1.XLC chart.

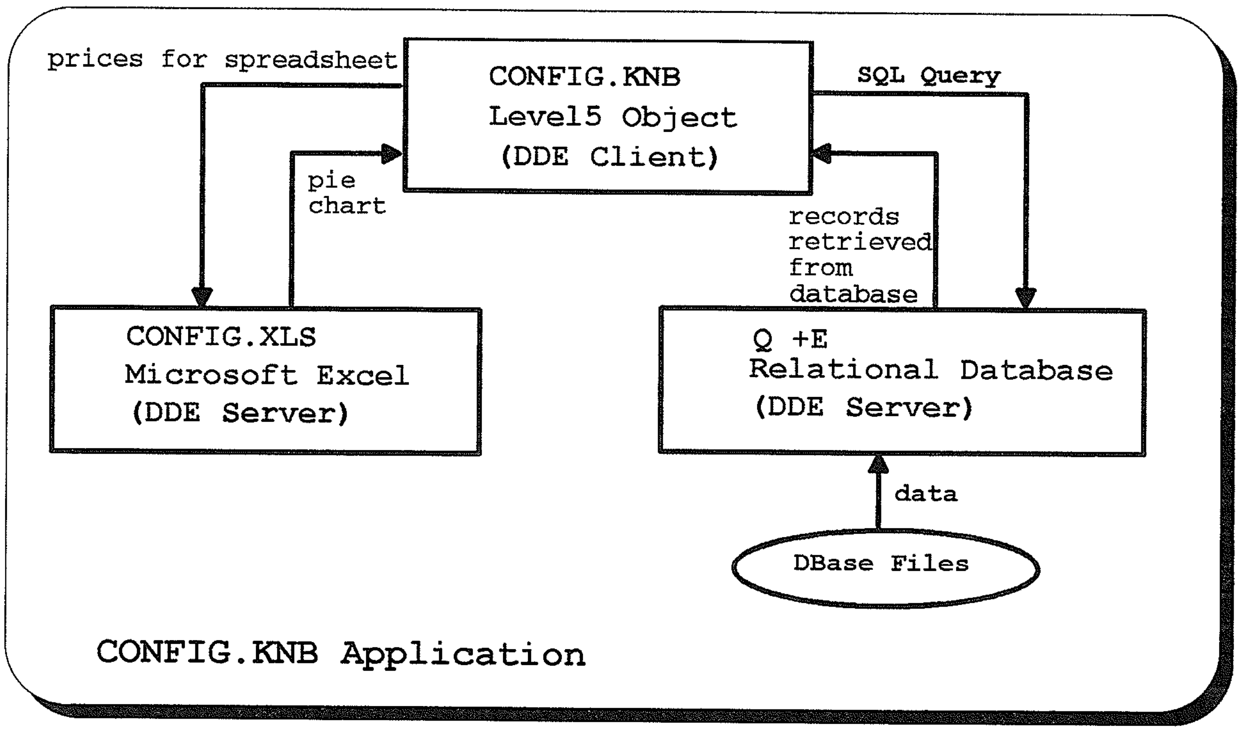
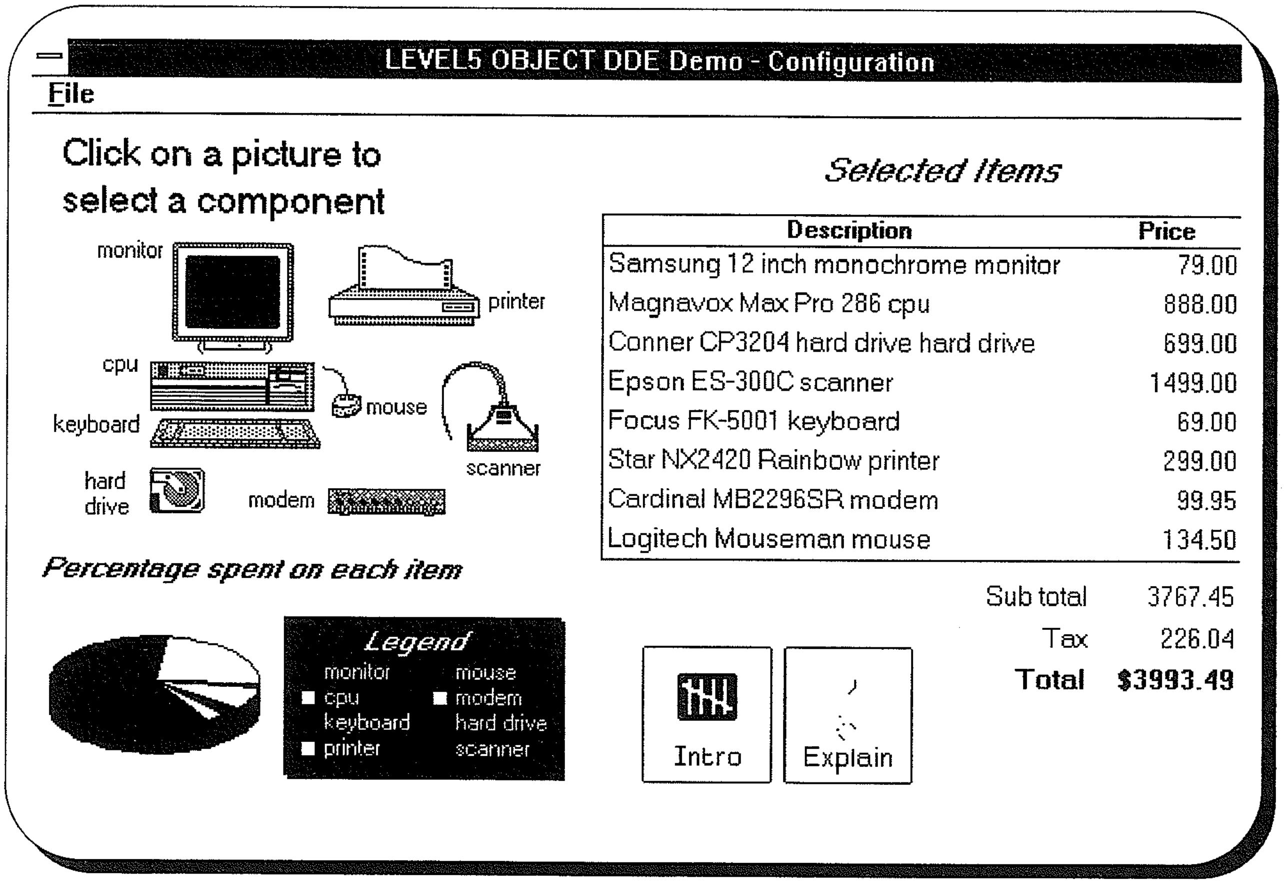


Figure 6.1

Both DDE instances are shown below:

```
INSTANCE open chart file ISA DDE
 WITH app := "EXCEL"
 WITH topic := "SYSTEM"
 WITH item := "[OPEN(\"CONFIG1.XLC\")]"
 WITH active := FALSE
 WITH action IS execute
 WITH link IS cold
 WITH append := TRUE
 WITH autostart := TRUE
 WITH time out := 0 00:00:10.000
 WITH default error handling := FALSE

INSTANCE open spreadsheet file ISA DDE
 WITH app := "EXCEL"
 WITH topic := "SYSTEM"
 WITH item := "[OPEN(\"CONFIG.XLS\")]"
 WITH active := FALSE
 WITH action IS execute
 WITH link IS cold
 WITH append := TRUE
 WITH autostart := TRUE
 WITH time out := 0 00:00:10.000
 WITH default error handling := FALSE
```



Selected Items

Description	Price
Samsung 12 inch monochrome monitor	79.00
Magnavox Max Pro 286 cpu	888.00
Conner CP3204 hard drive hard drive	699.00
Epson ES-300C scanner	1499.00
Focus FK-5001 keyboard	69.00
Star NX2420 Rainbow printer	299.00
Cardinal MB2296SR modem	99.95
Logitech Mouseman mouse	134.50
Sub total	3767.45
Tax	226.04
Total	**$3993.49**

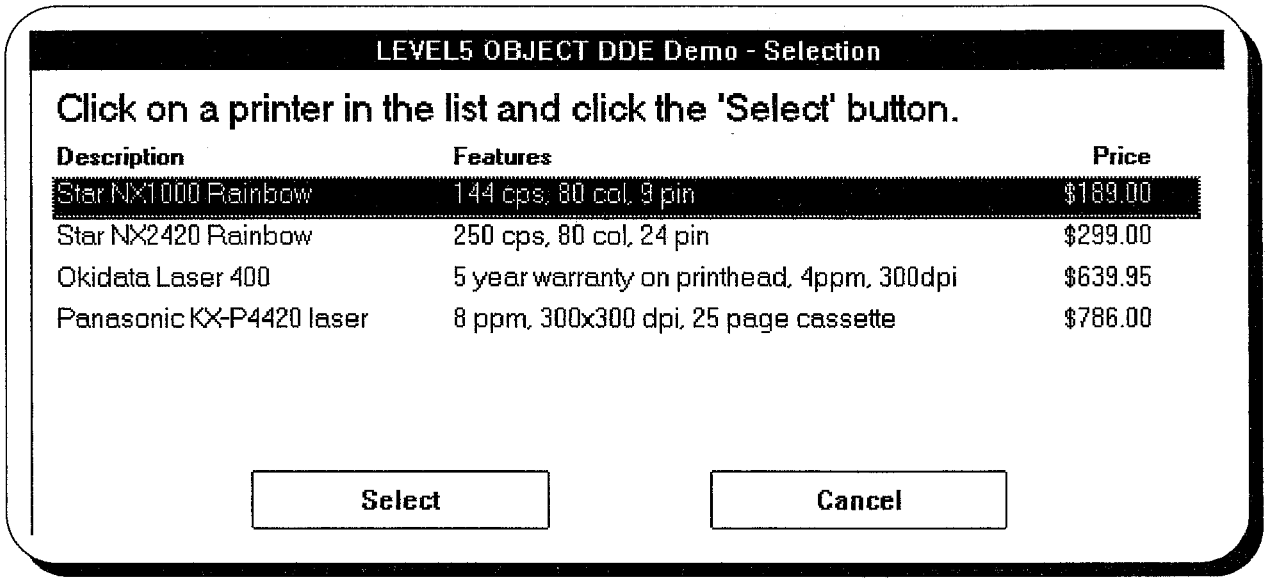
Figure 6.2

The application displays a graphic of a personal computer system consisting of several components (figure 6.2): a monitor, a CPU, a keyboard, etc. When a user clicks with the mouse on one of the components Level5 Object sends an SQL query to Q + E, for example:

```
SELECT item,features,price FROM invntry.dbf WHERE type = "monitor" ORDER BY price
```

Based on that query, Q + E retrieves several records from a database which is in DBase format. Retrieved records contain a description of the selected computer component, its features and a price. Q + E sends the retrieved records via DDE to Level5 Object. Level5 Object in turn displays the received records in a table asking the user to select one record (see figure 6.3). The selected item is placed in a Selected Items table displayed in the main window. The above sequence of events is accomplished by the following DDE system class instance:

```
INSTANCE select components ISA DDE
  WITH app := "QE"
  WITH item := "All"
  WITH active := FALSE
  WITH attachment := result table
  WITH action IS request
  WITH link IS cold
  WITH append := FALSE
  WITH autostart := TRUE
  WITH time out := 0 00:01:30.000
  WITH default error handling := FALSE
```



LEVEL5 OBJECT DDE Demo - Selection

Click on a printer in the list and click the 'Select' button.

Description	Features	Price
Star NX1000 Rainbow	144 cps, 80 col, 9 pin	$189.00
Star NX2420 Rainbow	250 cps, 80 col, 24 pin	$299.00
Okidata Laser 400	5 year warranty on printhead, 4ppm, 300dpi	$639.95
Panasonic KX-P4420 laser	8 ppm, 300x300 dpi, 25 page cassette	$786.00

Figure 6.3

The following When-Changed Method[1] is invoked when the user clicks on a component. This method creates and sends the actual SQL query to Q + E:

```
WHEN CHANGED
  BEGIN
    topic OF select components := CONCAT( "SELECT  item,features,price FROM
                                  invntry.dbf WHERE type = '", CONCAT( name OF
                                  component type, "' ORDER BY price"))
    active OF select components := TRUE
  ... END
```

[1] When-Changed Methods are user-defined methods (procedures) called by Level5 whenever a value of an attribute changes.

Level5 Object then sends the prices of all items from the Selected Items table to Microsoft Excel using the following DDE system class instance:

```
INSTANCE send prices ISA DDE
 WITH app := "EXCEL"
 WITH topic := "CONFIG.XLS"
 WITH item := "prices"
 WITH active := TRUE
 WITH attachment := current item price OF component type
 WITH action IS poke
 WITH link IS cold
 WITH append := TRUE
 WITH autostart := TRUE
 WITH time out := 0 00:01:00.000
 WITH default error handling := FALSE
```

Level5 Object then requests a pie chart from Microsoft Excel showing the percentage of each component's price in the total system price (see figure 6.2). This is accomplished by:

```
INSTANCE get chart ISA DDE
 WITH app := "EXCEL"
 WITH topic := "CONFIG1.XLC"
 WITH item := "chart"
 WITH active := TRUE
 WITH attachment := picture OF chart picture
 WITH link IS cold
 WITH append := TRUE
 WITH autostart := TRUE
 WITH time out := 0 00:01:30.000
 WITH default error handling := FALSE
```

The pie chart received from Microsoft Excel is displayed in the main window (see figure 6.2). All DDE links in this application are Cold Links meaning the **active** attribute of the **DDE** instance must be set each time to invoke a data or command exchange.

The above example shows several important capabilities of Level5 Object's DDE implementation. First, Level5 Object can conduct a DDE dialog with several applications simultaneously. Secondly, Level5 Object can transfer data in both text and graphic format. It also shows that data from a database can be accessed using DDE provided there is a server database application supporting DDE. Actually, DDE provides a flexible enough mechanism to make up for the lack of both a built-in database interface and the ability to call external functions.

6.2.2. DDE.KNB Application

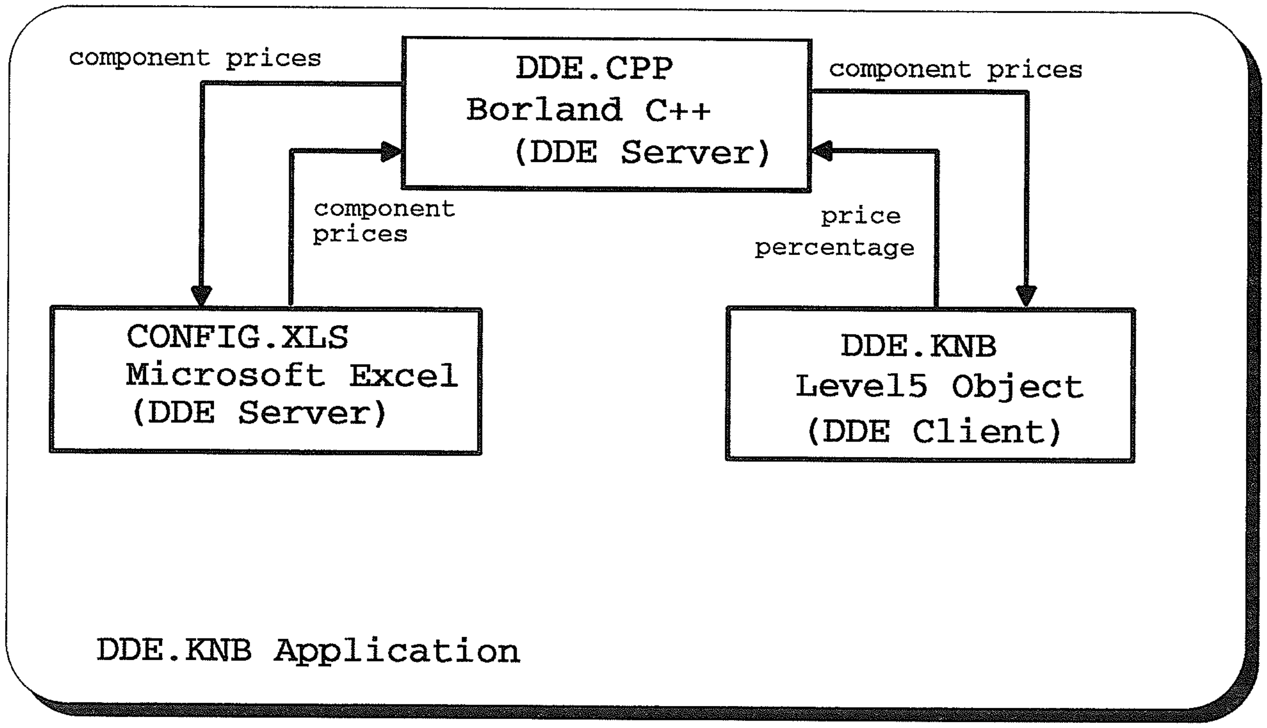


Figure 6.4

One of the limitations of Level5 Object's DDE capabilities is lack of support for DDE Server services. This means that Level5 Object applications can only take the role of a client in DDE conversations. In many cases we may want to develop a main application using a conventional programming language and have a Level5 Object application be a server to our main program. The main program would provide the user interface and other "conventional" functionality and it would ask the Level5 Object-based application (knowledge base) to do some processing whenever appropriate.

This example (figure 6.4) shows how to create a system in which a Level5 Object-based application DDE.KNB, even though a DDE client, acts as if it were a server with respect to a DDE.CPP program developed in Borland's C++.

The main program starts by opening a main window and launching two applications: Microsoft Excel and Level5 Object. After starting Microsoft Excel, DDE.CPP initiates a DDE conversation with Excel where DDE.CPP becomes a DDE client and Excel becomes a DDE server. The **topic** of the DDE conversation is System and then DDE.CPP sends a **DDE command** asking Microsoft Excel to load a spreadsheet called COMPUTER.XLS. After that DDE.CPP terminates the DDE conversation with Excel and starts a new one with a topic now being the spreadsheet name. This establishes a hot link between Excel and DDE.CPP. The following C++ code accomplishes this task:

```
if (WinExec(msgbuf, SW_SHOWMINIMIZED) <= 32)
{
    ok = 0;
   MessageBox( HWindow,"Could not start Excel.","Error",
                  MB_ICONEXCLAMATION | MB_OK );
}

if (ok)
```

```
{
   InitiateDDE( "EXCEL", "System" );
   SendCommand("[OPEN(\"computer.xls\")]");
   TerminateDDE();
   InitiateDDE( "EXCEL", "computer.xls" );
   InitHotLinks();
}
```

DDE.CPP starts Level5 Object with the following command:

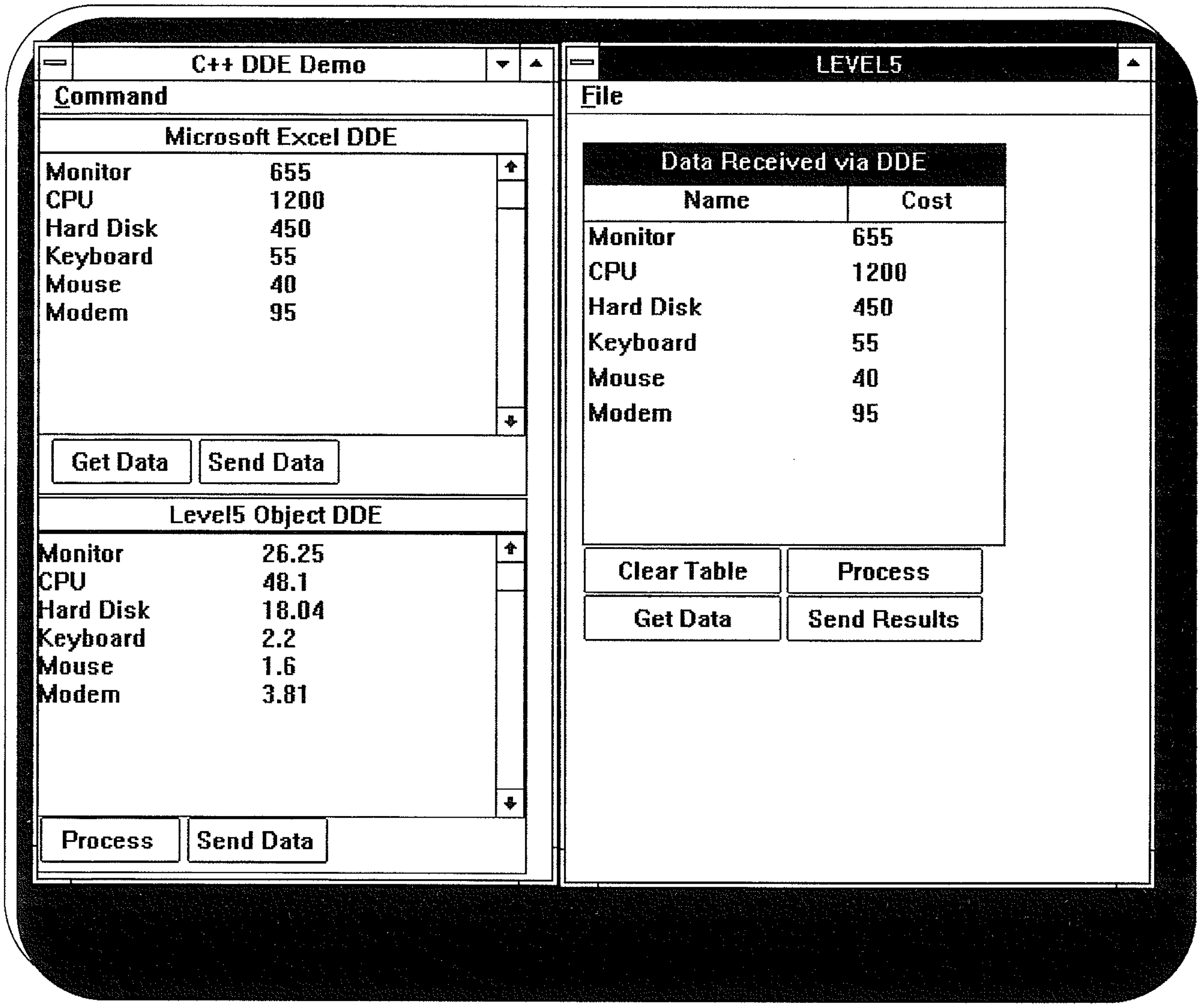
```
WinExec("c:\\l5o25\\l5.exe c:\\prg\\l5o\\dde\\dde.knb /r", SW_SHOWMINIMIZED);
```

This command starts Level5 Object which automatically loads and starts running the DDE.KNB knowledge base. DDE.KNB immediately initiates two DDE conversations with DDE.CPP using the following DDE system class instances:

```
INSTANCE dde1 ISA DDE
  WITH app := "DDE"
  WITH topic := "LEVEL5"
  WITH item := "DATA"
  WITH active := FALSE
  WITH attachment := DataReceived
  WITH data ready := FALSE
  WITH link IS hot
  WITH append := FALSE
  WITH autostart := FALSE
  WITH time out := 0 00:00:10.000
  WITH default error handling := TRUE

INSTANCE dde2 ISA DDE
  WITH app := "DDE"
  WITH topic := "LEVEL5"
  WITH item := "CONTROL"
  WITH active := FALSE
  WITH attachment := str OF ControlStr1
  WITH data ready := FALSE
  WITH link IS warm
  WITH append := FALSE
  WITH autostart := FALSE
  WITH time out := 0 00:00:10.000
  WITH default error handling := TRUE
```

The **dde1** instance initiates a hot DATA link and the **dde2** instance initiates a warm CONTROL link. DATA link is used to exchange data in both directions while CONTROL link is used by DDE.CPP to send commands to DDE.KNB. The CONTROL link then enables a DDE server to send commands to a DDE client so it appears as if their roles have been reversed.

After starting both Excel and Level5 Object, DDE.CPP waits for the user to select an action. The main window (C++ DDE DEMO) contains two child windows (see figure 6.5), the upper one is responsible for the DDE dialog with Excel while the lower one controls the DDE dialog with the Level5 Object DDE.KNB knowledge base. When the user clicks on the Get Data button in the Microsoft Excel DDE window, DDE.CPP sends a DDE request for data to Microsoft Excel. Microsoft Excel responds by retrieving the requested data from the previously open spreadsheet and sending it back to DDE.CPP. DDE.CPP, after receiving data from Excel, displays the data in the window. The user can modify that data and send it back to Excel. The spreadsheet can be modified by clicking on the Send Data button. Data received from Excel can be sent to DDE.KNB by clicking on the Send Data button in the Level5 Object DDE window.



Monitor	655
CPU	1200
Hard Disk	450
Keyboard	55
Mouse	40
Modem	95

Monitor	26.25
CPU	48.1
Hard Disk	18.04
Keyboard	2.2
Mouse	1.6
Modem	3.81

Name	Cost
Monitor	655
CPU	1200
Hard Disk	450
Keyboard	55
Mouse	40
Modem	95

Figure 6.5 DDE.CPP and DDE.KNB running simultaneously (Excel in the background)

DDE.KNB receives data from DDE.CPP via the hot DATA link which is represented by **dde1** instance of the DDE system class. Immediately after data is received a table in the main window is updated. Clicking on the Process button in the DDE.KNB window will trigger processing which in this case calculates the percentage of a total computer price for each item. The results of the processing are sent back to DDE.CPP via the same DATA link by pressing on the Send Results button.

This is accomplished by the following Level5 Object's code:

```
ATTRIBUTE SendData SIMPLE
 WHEN CHANGED
  BEGIN
   active OF dde1 := FALSE
   attachment OF dde1 := Result
   action OF dde1 IS poke := TRUE
   active OF dde1 := TRUE
   attachment OF dde1 := DataReceived
  END
```

In this case DDE.KNB simply uses the capabilities of a DDE client. The same result can be achieved by clicking on the Process button of DDE.CPP. Here all actions are triggered by DDE.CPP even though it is a DDE server. This is achieved via the CONTROL warm link. DDE.CPP sends a command "GO" in the string format to DDE.KNB. Since this is a warm link **data ready OF dde2** becomes TRUE which triggers the attached demon:

```
DEMON 2
IF data ready OF dde2
THEN getControlStr := TRUE
AND FORGET  data ready OF dde2
```

This in turn causes the following When-Changed method to be invoked:

```
WHEN CHANGED
 BEGIN
  str OF ControlStr1 := ""
  action OF dde2 IS request := TRUE
  action OF dde2 IS request := FALSE
  IF str OF ControlStr1 = "go" THEN
   BEGIN
    DoProcess := TRUE
    SendData := TRUE
    str OF ControlStr1 := ""
    ASK display3
   END
 END
```

As a result DDE.KNB performs the processing and sends the results to DDE.CPP using the DATA DDE link. Using the same mechanism other commands could be sent to DDE.KNB requesting other services. Consequently, the Level5 Object application acts as a server to DDE.CPP even though it is actually a client in the DDE protocol. The main advantage of having a Level5 Object application act as a server to a conventional application is that the expert system can be hidden from the user. The expert system will be virtually embedded in the conventional system. The user will perceive the new system as the improved conventional application so he or she will fill comfortable using it.

6.3. Dynamic Data Exchange in Kappa-PC

Kappa-PC provides full support of DDE services including both client and server functionality [*Kappa-PC User's Guide*, 1992]. As a client Kappa-PC can send DDE messages to other applications requesting data, modifying data, or executing commands remotely. This functionality is implemented by four KAL (Kappa-PC Application Language) functions: **RemoteCheckStatus**, **RemoteExecute**, **RemoteGet**, and **RemoteSet**. As a server Kappa-PC responds to DDE messages from other applications. For example, the following DDE command sent to Kappa-PC from any application will cause Kappa-PC to open a window with the "Hello" message: **ExecuteRemote** "PostMessage (""Hello"");" application Kappa.

By default, the KAL commands are synchronous, but they can be made asynchronous by specifying the optional argument NOWAIT with the DDE commands. Synchronous DDE functions do not time out. They wait until the server application responds. While waiting, they release control to other applications, thus allowing input from those applications. On the other hand, asynchronous DDE functions achieve executions in series [*Kappa-PC User's Guide*,1992]. The four DDE KAL functions are also supported through the C library interface:

- **BOOL KpcCheckStatus(int iDDEJobId)**
- **int KpcRemoteExecute(BOOL bWait, LPSTR sCommand, LPSTR sApp, LPSTR sTopic)**
- **int KpcRemoteGet(BOOL bWait, LPSTR sCommand, LPSTR sApp, LPSTR sTopic, OBJECTID idObj, ATOMID idSlot)**
- **int KpcRemoteSet(BOOL bWait, LPSTR sCommand, LPSTR sApp, LPSTR sTopic, LPSTR sValue, LPSTR sTopic)**

The functionality of the commands is identical to the KAL version. The arguments are not the same, however. bWait and sTopic arguments, unlike in the KAL version, are required. Implementation of the DDE protocol in Kappa PC is flexible enough to satisfy the needs of most applications. It is also easy to use, since Kappa PC hides some lower-level details, for example the process of DDE initialization and the DDE link types.

6.4. Summary

The DDE implementation in Level5 Object is quite complete except that Level5 Object can only assume a role of a client and cannot be a server. This may be a serious limitation if we want to have a main program written in conventional programming language and Level5 Object to be called only to process a knowledge base and return the results. As shown in the second example there is a way around this limitation - Level5 Object being a DDE client acts as if it were a server, nevertheless a DDE server support by Level5 Object would be a valuable improvement[1].

Support for a client-server architecture, implemented as a DDE protocol, offers the most powerful and most flexible method of integrating an expert system technology with conventional technology. It can provide the functionality of all other discussed methods: database interface, external function calls, and embedding in a conventional application. Despite its power and flexibility DDE is relatively easy to use which makes the learning curve very short.

1 Server support is planned for version 3.0 of Level5 Object to be released in the fall of 1993.

7. Knowledge Base Management

Knowledge base management facilities become increasingly important when a knowledge base becomes larger and more complex. They are absolutely crucial for successful implementation of a real-world expert system containing hundreds of objects and thousands of rules. Most commercial expert system tools provide some of those utilities but none provides all of them. Most important knowledge base management facilities are described below:

- **Object Editors and Rule Editors**: easy-to-use editors that allow a developer to view, modify, and delete classes and objects, rules, and methods attached to those rules and objects;

- **Object and Rule Browsers**: allow a developer to display a hierarchy of objects showing their relationships, inheritance, attributes, and attached methods, as well as relationships among rules (or a knowledge tree); it should be possible to invoke editors directly from the object or rule browser just by clicking with a mouse on a desired element;

- **Debugger**: allows stepping through rules and methods, setting break-points, and monitor values of object attributes and variables;

- **Session Trace or History facility**: records in a text file all activities (activated rules, changes in values of attributes, methods called, etc.) of an expert system during a session;

- **Explanation facility**: allows both a developer and the end-user to find out how a system reached a conclusion or why it did not reach it;

- **Knowledge Structuring facilities**: ability to structure knowledge into small, independent knowledge bases (or rule groups) that can perform independent tasks (e.g. solve subproblems); it should be possible to switch the processing between those knowledge bases (both chaining and calling); meta-knowledge might be necessary to control switching between knowledge bases;

- **Knowledge Sharing facilities**: ability to store pieces of knowledge, both objects with attached methods and rules, in disk files and import them to other knowledge-based applications.

7.1. Knowledge Management Facilities in Level5 Object

Following is a discussion of knowledge management facilities found in Level5 Object [*Level5 Object User's Guide,* 1990].

7.1.1. Object Editors and Rule Editors

Level5 Object provides easy-to-use and well-designed editors for the following elements of a knowledge base: rules (fig. 7.3), objects and attributes(fig. 7.2), agenda (fig. 7.4), methods (fig. 7.5), screen forms (fig. 7.6). All editors are well integrated and easily accessed from menus, the icon bar (fig. 7.1), or other parts of the system. For example, by double-clicking on a rule displayed in a knowledge tree it is possible to display that rule in a rule editor in order to view or modify it.

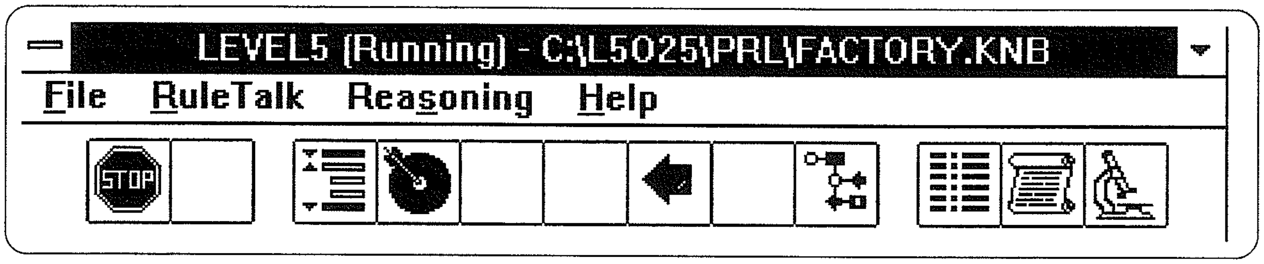
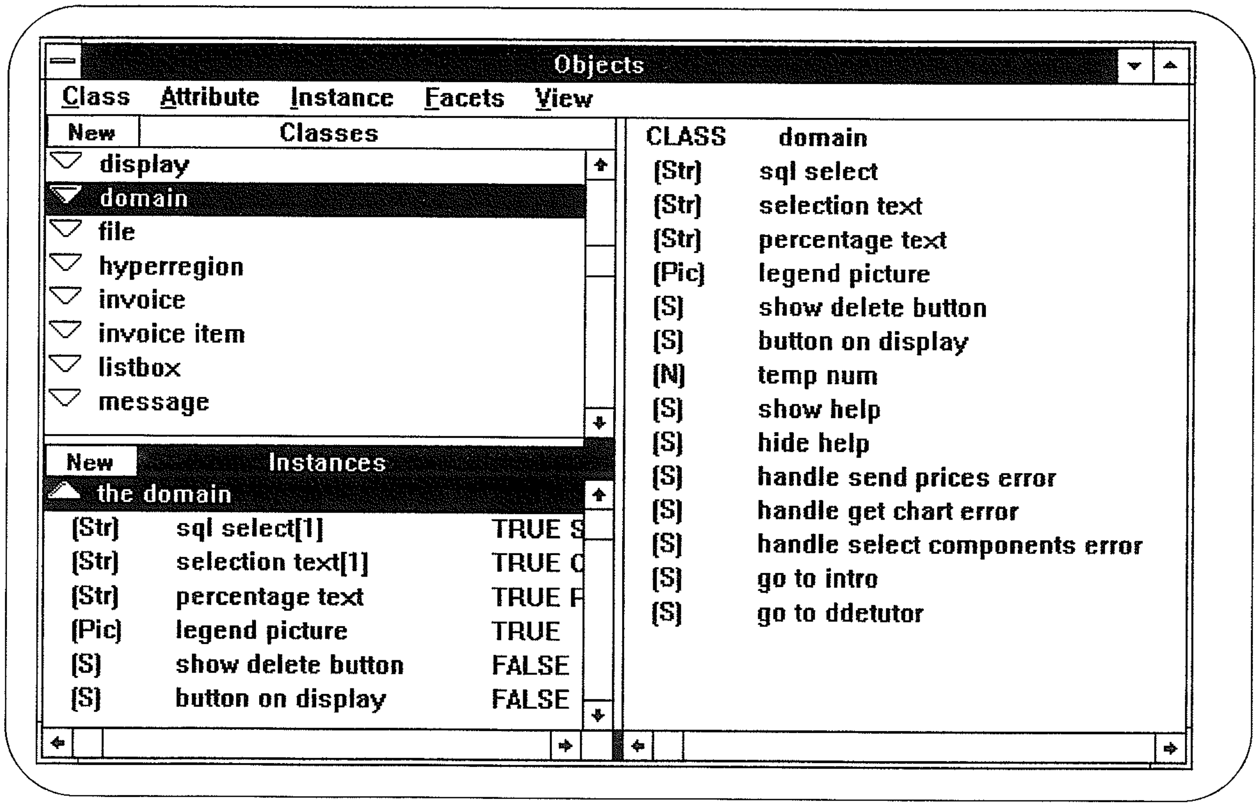


Figure 7.1 Icon Bar (Tool Bar)



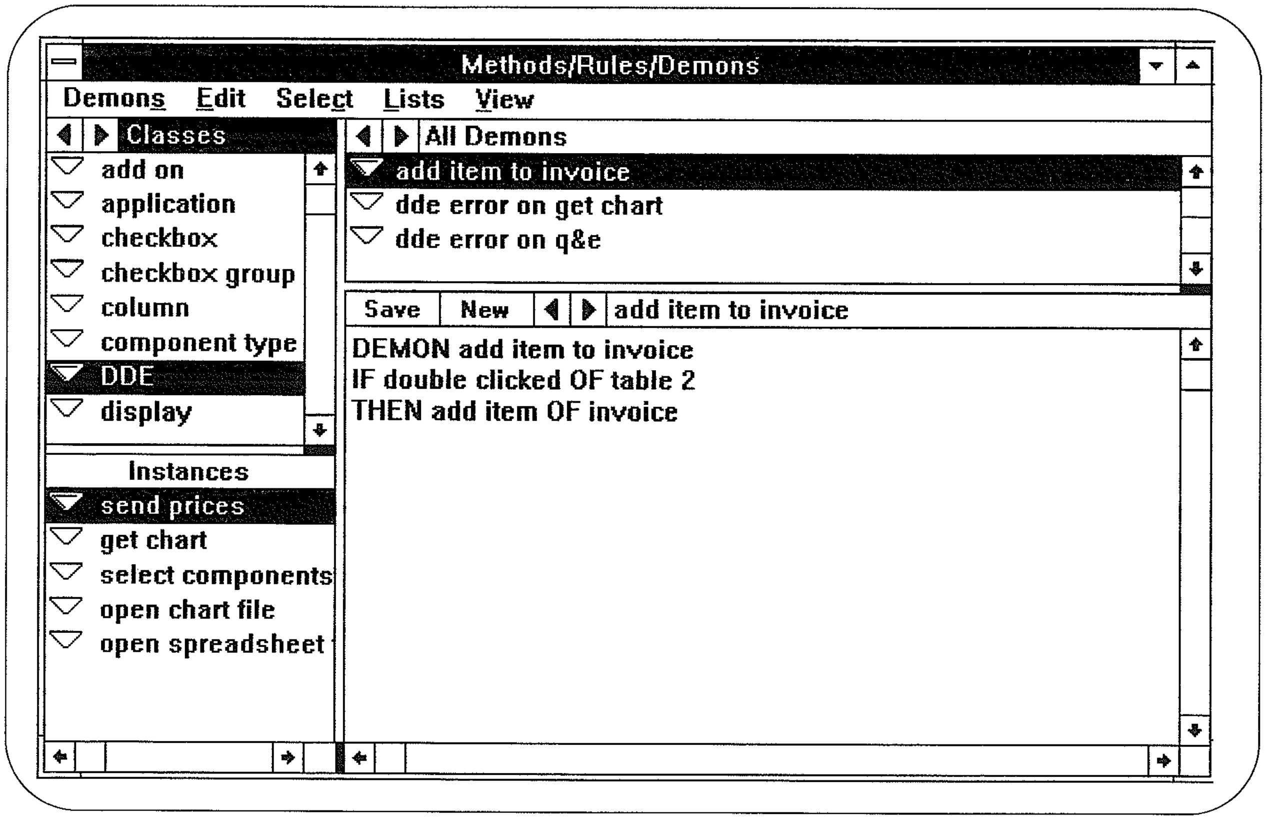
display
domain
file
hyperregion
invoice
invoice item
listbox
message

the domain

[Str]	sql select[1]	TRUE S
[Str]	selection text[1]	TRUE C
[Str]	percentage text	TRUE F
[Pic]	legend picture	TRUE
[S]	show delete button	FALSE
[S]	button on display	FALSE

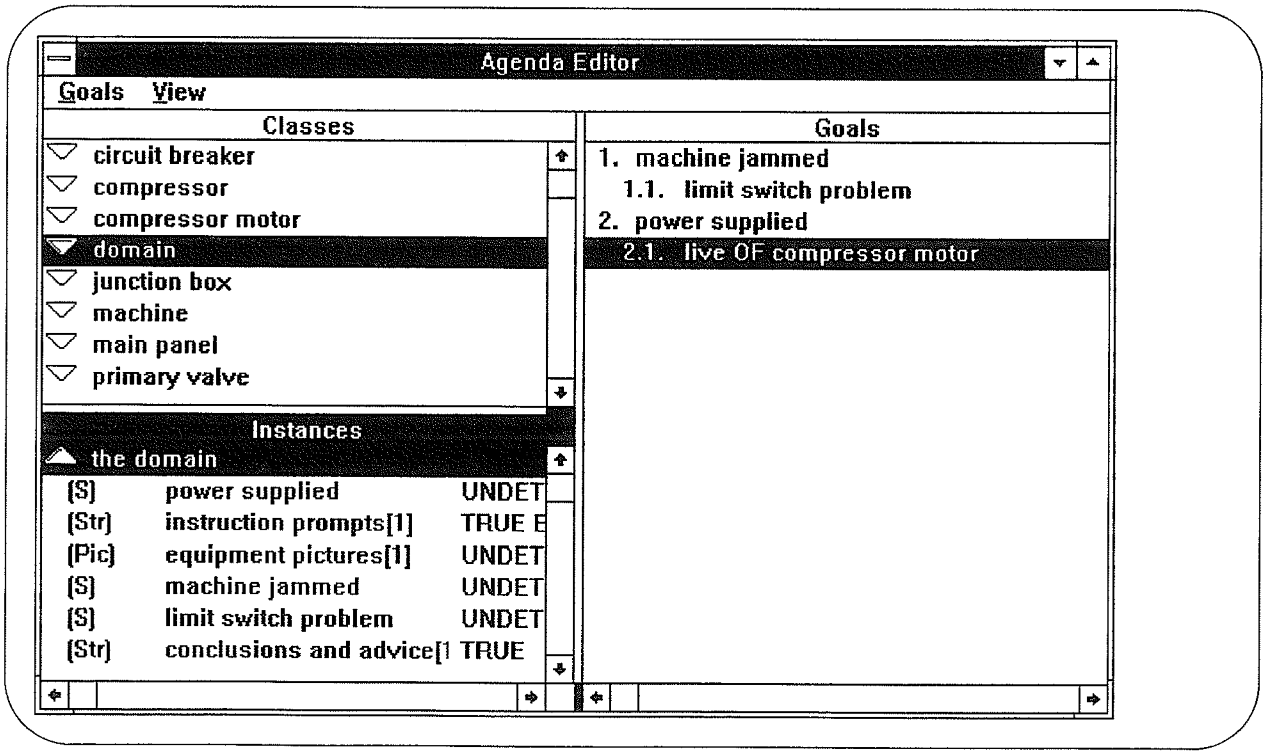
CLASS	domain
[Str]	sql select
[Str]	selection text
[Str]	percentage text
[Pic]	legend picture
[S]	show delete button
[S]	button on display
[N]	temp num
[S]	show help
[S]	hide help
[S]	handle send prices error
[S]	handle get chart error
[S]	handle select components error
[S]	go to intro
[S]	go to ddetutor

Figure 7.2 Objects/Attributes Editor



```
DEMON add item to invoice
IF double clicked OF table 2
THEN add item OF invoice
```

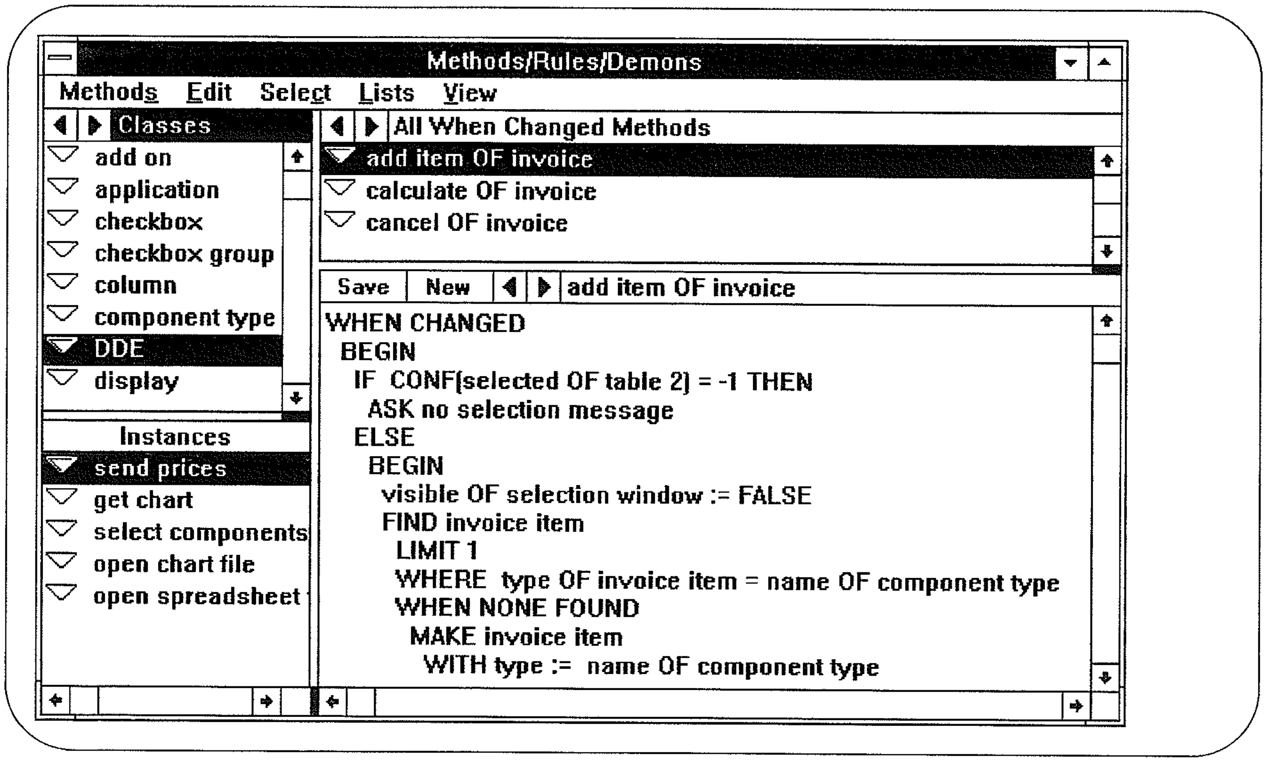
Figure 7.3 Rules/Demons Editor



```
[S]     power supplied              UNDET
[Str]   instruction prompts[1]      TRUE E
[Pic]   equipment pictures[1]       UNDET
[S]     machine jammed              UNDET
[S]     limit switch problem        UNDET
[Str]   conclusions and advice[1    TRUE
```

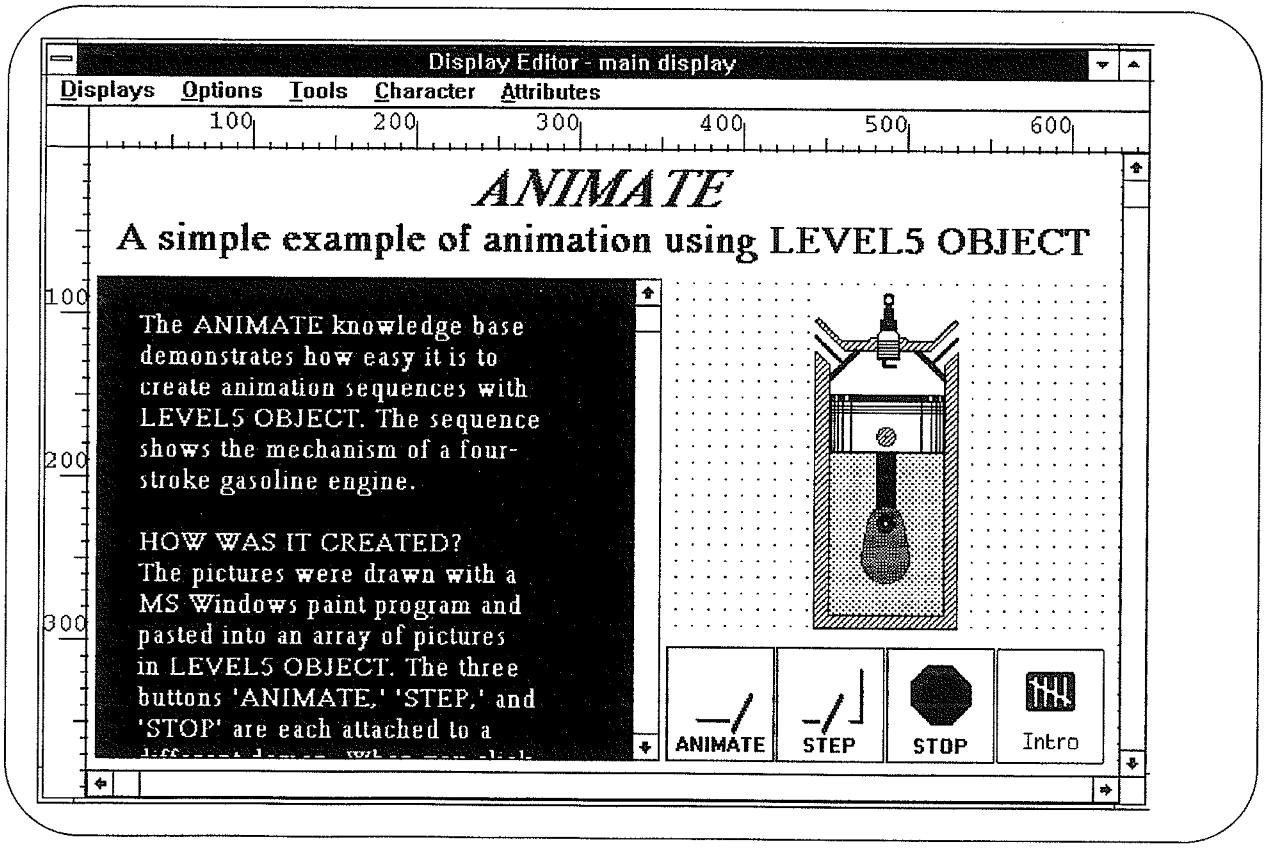
```
1.  machine jammed
  1.1.  limit switch problem
2.  power supplied
  2.1.  live OF compressor motor
```

Figure 7.4 Agenda Editor



```
WHEN CHANGED
 BEGIN
  IF  CONF[selected OF table 2] = -1 THEN
   ASK no selection message
  ELSE
   BEGIN
    visible OF selection window := FALSE
    FIND invoice item
     LIMIT 1
     WHERE  type OF invoice item = name OF component type
     WHEN NONE FOUND
      MAKE invoice item
       WITH type :=  name OF component type
```

Figure 7.5 Methods Editor


Display Editor - main display
Displays Options Tools Character Attributes
100
200
300
400
500
600
ANIMATE
A simple example of animation using LEVEL5 OBJECT
ANIMATE
STEP
STOP
Intro

The ANIMATE knowledge base demonstrates how easy it is to create animation sequences with LEVEL5 OBJECT. The sequence shows the mechanism of a four-stroke gasoline engine.

HOW WAS IT CREATED?
The pictures were drawn with a MS Windows paint program and pasted into an array of pictures in LEVEL5 OBJECT. The three buttons 'ANIMATE,' 'STEP,' and 'STOP' are each attached to a

Figure 7.6 Display Editor

7.1.2. Object and Rule Browsers

Level5 Object provides the Knowledge Tree (fig. 7.7) facility which shows the relationships among rules and demons. The tree can be displayed in several formats (e.g. collapsed, expanded, partially expanded), it can show all rules, only backward-chaining rules, or only demons. It can also show which rules where invoked during a session. Although the Knowledge Tree can be useful in some situations, it is difficult to follow in the case of large knowledge bases. There is no object browser that would show relationships between objects.

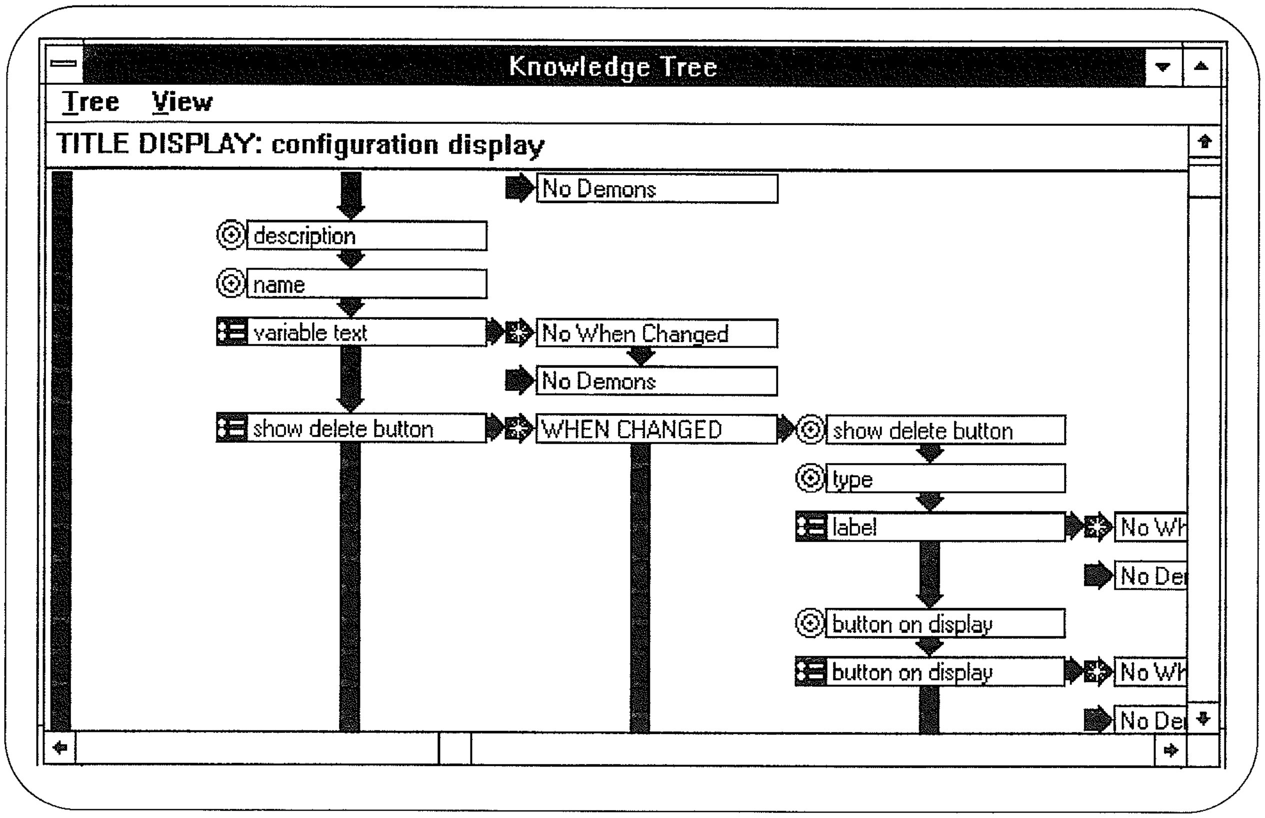
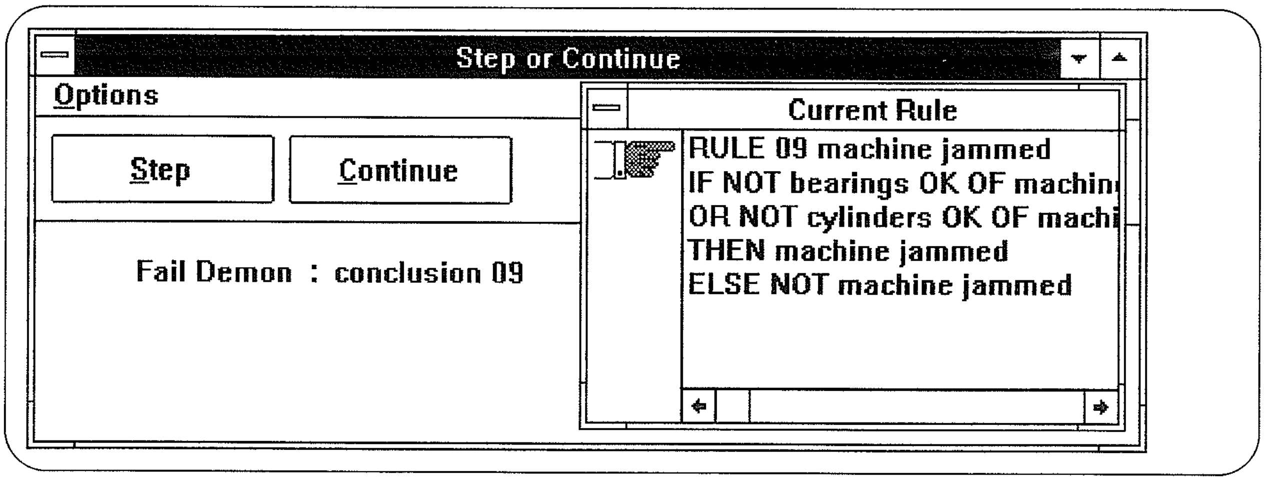


Figure 7.7 Knowledge Tree

7.1.3. Debugger

Level5 Object is equipped with a debugger (fig. 7.8). It allows setting break points on rules or attributes and it supports stepping through a knowledge base. The debugger always steps into a method, it does not allow to go through a method in one step like most traditional debuggers do. It also does not show which instruction or line inside a method is being executed. When in the stepping mode the debugger allows to examine values of attributes through a Value List utility.



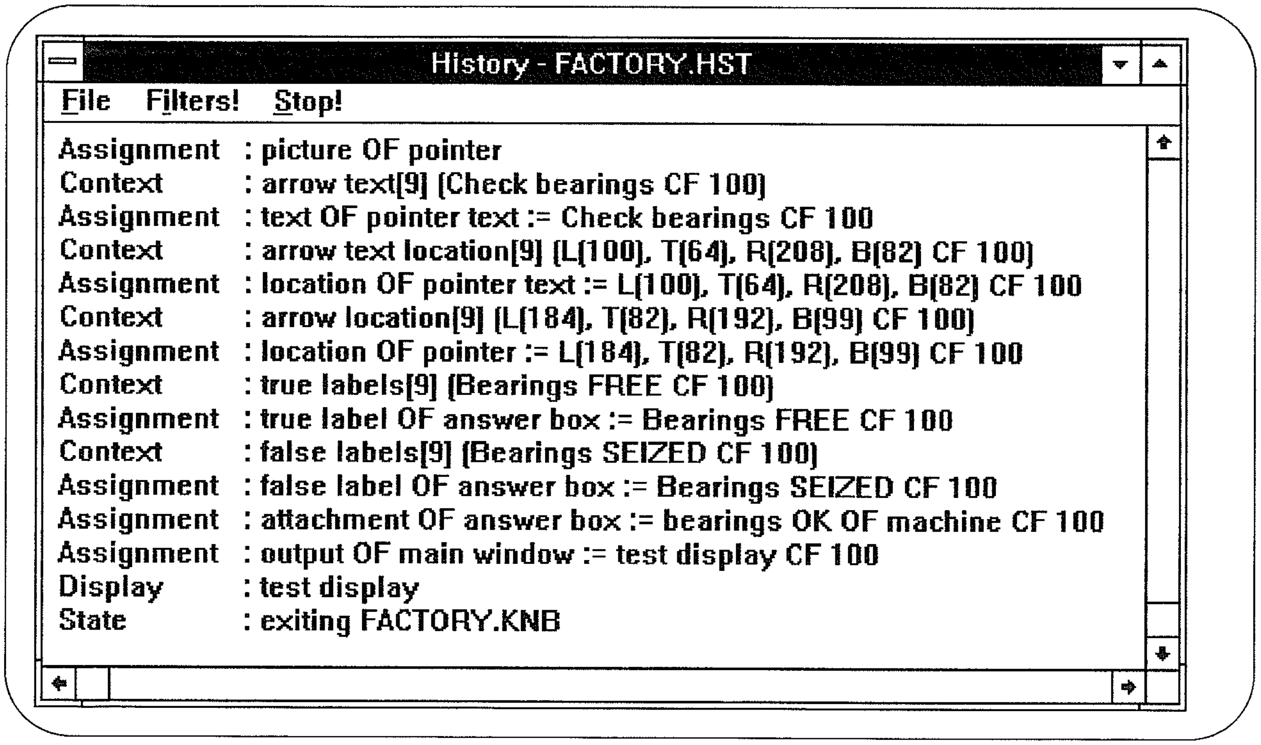
Fail Demon : conclusion 09

```
RULE 09 machine jammed
IF NOT bearings OK OF machin
OR NOT cylinders OK OF machi
THEN machine jammed
ELSE NOT machine jammed
```

Figure 7.8 Debugger

7.1.4. Session Trace or History facility

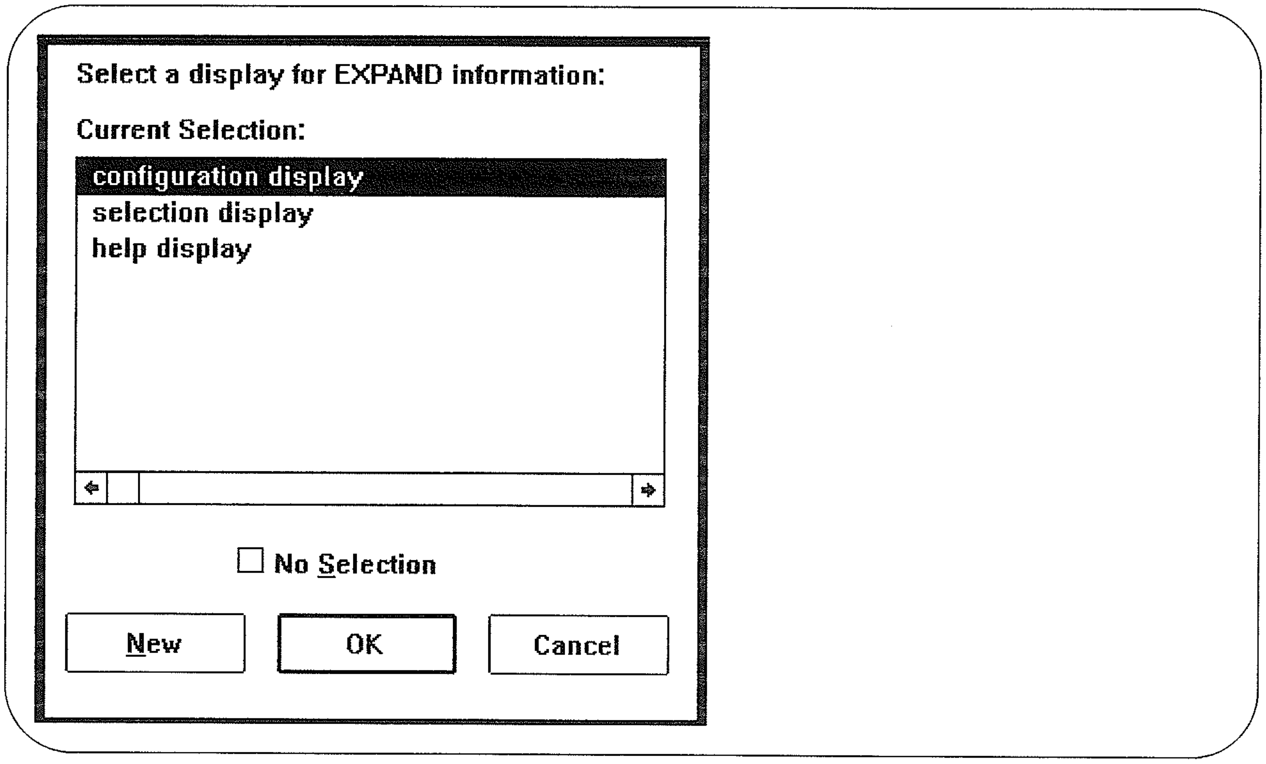
Level5 Object provides a history facility (fig. 7.9). It allows a user to store a history of a session to a text file in a readable format. The history shows all actions performed by Level5 Object's inference engine. Also, if a user is not interested in recording all the actions, he or she can set filters that filter out unwanted information. The history facility is a very useful debugging and testing tool.



```
Assignment  : picture OF pointer
Context     : arrow text[9] (Check bearings CF 100)
Assignment  : text OF pointer text := Check bearings CF 100
Context     : arrow text location[9] (L(100), T(64), R(208), B(82) CF 100)
Assignment  : location OF pointer text := L(100), T(64), R(208), B(82) CF 100
Context     : arrow location[9] (L(184), T(82), R(192), B(99) CF 100)
Assignment  : location OF pointer := L(184), T(82), R(192), B(99) CF 100
Context     : true labels[9] (Bearings FREE CF 100)
Assignment  : true label OF answer box := Bearings FREE CF 100
Context     : false labels[9] (Bearings SEIZED CF 100)
Assignment  : false label OF answer box := Bearings SEIZED CF 100
Assignment  : attachment OF answer box := bearings OK OF machine CF 100
Assignment  : output OF main window := test display CF 100
Display     : test display
State       : exiting FACTORY.KNB
```

Figure 7.9 History Facility

7.1.5. Explanation facility


Select a display for EXPAND information:

Current Selection:

configuration display
selection display
help display

☐ No Selection

Figure 7.10 Expand Facility

Level5 Object offers explanation facility through the Expand facet that can be attached to attributes (fig. 7.10). The Expand facet associates an attribute with a display window that can contain both textual and graphical explanation of this attribute, for example it can explain why the system is trying to determine a value of the attribute. The ability to use graphics as part of an explanation significantly improves the quality of the information presented to the end user.

7.1.6. Knowledge Structuring facilities

Level5 Object allows chaining from one knowledge base to another. Attribute values can be passed to the next (in chain) knowledge base through the Share attribute facet. Although the concept of chaining is useful, the implementation makes it virtually useless because of a very long execution time. Also, it would be useful to be able to call a knowledge base, and then to return back to the previous knowledge base. Knowledge base context would have to be preserved and restored automatically by Level5 Object.

Example of chaining:

```
WHEN CHANGED
 BEGIN
  IF  go = TRUE THEN
    CHAIN "myapp.knb"
 END
```

7.1.7. Knowledge Sharing facilities

Level5 Object allows exporting a knowledge base to a text file. The knowledge base is stored in a readable text format - it is translated to Level5 Object's PRL (Production Rule Language). It is also possible to import a previously exported knowledge base. This allows developers to port

Level5 Object-based applications to other hardware platforms. However, export and import facilities work only with entire knowledge bases only. It would be useful to be able to export/import parts of a knowledge base, for example a rule or an object. An example PRL code is shown below:

```
$VERSION25
$LOCATIONS ARE PIXELS

ATTRIBUTE str1 STRING
ATTRIBUTE str2 STRING
ATTRIBUTE convert SIMPLE
 WHEN CHANGED
  BEGIN
   ACTIVATE  "IPU,SERVER,C:\PRG\L5O\SERVER\TOUPPER.EXE"
    SEND  str1
    RECEIVE  str2
  END

INSTANCE the application ISA application
 WITH unknowns fail := TRUE
 WITH threshold := 50
 WITH title display := display 1
 WITH ignore breakpoints := FALSE
 WITH reasoning on := FALSE
 WITH numeric precision := 8
```

7.2. Summary of Knowledge Management Facilities in Level5 Object

Level5 Object offers a very high-quality development tools when compared with other expert system packages, for example Kappa-PC. It provides a wide range of tools which should satisfy most developers. Level5 Object lacks however some important features and some tools could be improved. One of the most useful tools lacking in Level5 Object is an Object Browser that would graphically show dependencies (inheritance relations) between objects. The Knowledge Tree tool should be improved mainly in the area of the display formats. The

Debugger should point to individual statements inside methods. It should be possible to export/import objects like it is done in Kappa-PC. Chaining to other knowledge bases should be more efficient and it should be possible to call other knowledge bases in the same way as procedures can be called in conventional programming languages. It should be possible to put comments into all methods, rules and attributes. A very useful tool would be a knowledge base compiler that could generate C code like it is done in Kappa-PC. This would allow a developer to create very efficient delivery version of a knowledge based system after development is completed.

8. Conclusions

Expert Systems of the early eighties were mostly stand-alone and isolated from the rest of computing environment. This was a direct result of an AI-centric view represented by most researchers in Artificial Intelligence [*Earl D. Sacerdoti,* 1989]. Despite a great success of those early expert systems it quickly became clear that isolated expert systems could not satisfy the needs of complex computing systems. The majority of tasks in a real-world computing environment can be solved by conventional-computing technologies. Therefore, if AI systems are to become a real part of computing environments they must integrate well with them. An ideal knowledge-based tool should have the following features:

- should have access to all major database systems via SQL using a client-server architecture;
- should be able to call external programs/functions (regardless of the programming language they are written in) using the DLL mechanism and/or other methods provided by the operating system;
- should be able to exchange data and commands with other applications using the DDE protocol or other client/server communication mechanism provided by the operating system;
- should be embeddable in conventional programs using object-oriented programming technology and/or client/server architecture;
- should fully utilize object-oriented technology in all aspects of integration;
- should be able to run in distributed network environments and integrate with remote applications (running on other network nodes).

Many today's knowledge-based tools developers recognize the importance of integration with conventional-computing technologies and benefits that this integration can bring to knowledge-based systems. Most commercial knowledge-based tools, despite some limitations, evolve quickly in the direction of full integration. This process is also accelerated by a great progress in operating systems. New tools such as Dynamic Link Libraries, Dynamic Data Exchange, Object Linking and Embedding are now parts of MS Windows and OS/2. These tools make the integration much easier and more powerful. A relatively new SQL standard database interface makes it possible to develop knowledge-based tools with a generic database interface. It gives knowledge-based tools more power and flexibility and at the same time relieves the developers of knowledge-based tools from a difficult task of building their own database interfaces.

With new, object-oriented, operating systems such as Pink[1] on the horizon we can expect even better integration of knowledge-based systems with other components of a computing environment [*Thompson, T.*, 1993]. A new hardware platform, PowerPC[2], promises to create a new personal computer standard by unifying all major operating systems: Unix, Mac's Operating System, DOS, MS Windows, and OS/2. If this feat is accomplished we can expect knowledge-based systems integrating with other applications across different operating systems.

In the most recent years researchers started combining knowledge-based systems with new emerging software technologies such as neural networks, multimedia and virtual reality,

---

1 **Pink** is a new, object-oriented, operating system being jointly developed by IBM and Apple. It is scheduled for release in 1995.

2 **PowerPC** is a new, RISC-based, personal computer architecture being jointly developed by IBM, Apple, and Motorola. First PowerPC computers will be available at the end of 1993.

case based reasoning, and genetic algorithms. Although the hybrid systems are largely experimental, preliminary results indicate that such coupling can enhance problem-solving capabilities of knowledge-based systems [Hedberg, S., *New Knowledge Tools*, 1993].

Multimedia is a technology that allows combining graphic images, video movies, and sounds in software applications. Virtual reality is a technology which allows creating virtual models of various 3-D environments in a computer. A user of such a system can control objects in the 3-D space in such a way that he has the impression of this being real. Multimedia and virtual reality are exciting technologies by themselves, but by combining them with knowledge-based systems researchers are creating some new fascinating possibilities. A knowledge-based system coupled with multimedia and virtual reality will greatly improve the quality and power of a user interface. We can expect the systems guiding the users through the maze of menus and application features. Those new systems will be capable of intelligent storage, indexing, retrieval, and distribution of text, graphics, video clips, and sounds. An example of coupling virtual reality with knowledge-based technology would be an electronic shopping mall. A user could "walk" through the stores in 3-D space using a mouse or joystick. Intelligent agents would advice the user on, for instance, buying something or finding particular items [Hedberg, S., *See, Hear, Learn*, 1993].

Neural networks consist of parallel networks, or groups, of simple, highly interconnected processing units. They are well suited for pattern recognition, foreign language translation, process control, 3-D vision, and parallel implementations of routine processing tasks [*Liebowitz, J.*, 1993]. Neural networks offer great potential benefits to knowledge-based systems. Combining the two technologies will result in a new generation of self-adaptive, capable of learning, systems. One company, called Gensym (Cambridge, MA), introduced a new product called NeurOnLine. It layers neural-network technology onto G2 Real-Time Expert System (Gensym's general-purpose KBS/process-control tool). NeurOnLine's algorithms allow it to learn while it is monitoring a process. The result is a self-learning system that adapts to a changing process [Hedberg, S., *New Knowledge Tools*, 1993]. Combining neural networks and knowledge-based systems provides system improvements in many areas, including graceful system degradation, generalization, explicit and implicit reasoning, incremental learning, reliability, and flexibility. On the other hand, such hybrid systems are more complex, difficult to develop and maintain.

Case Based Reasoning (CBR) technology enables systems to store past experiences or situations as cases, analyze and process the data, and suggest ways of solving a problem based on those cases [*Liebowitz, J.*, 1993]. A CBR system has two primary components: a case base and a problem solver. A case base contains descriptions of previously solved and unsolved problems. A problem solver consists of a case retriever and a case reasoner. The case retriever identifies data in the case base that most closely fits the situation. The case reasoner examines the cases and, with the aid of domain knowledge, performs adaptation, synthesis, or prediction. CBR and rule-based systems complement one another. Rules handle large areas of problem domains well, but they are less useful and cost effective in boundary areas where subtle contexts tend to exist. Cases, on the other hand, can model entire domains if there is enough cases to cover all the problems. It is too expensive to cover an entire domain with cases and CBR systems tend to perform rather shallow reasoning. Therefore, the best solution is to model a domain with rules as far as possible, and then apply CBR technology to boundary regions.

Another emerging AI technology is genetic algorithms - adaptive, general-purpose search techniques based on the principles of population genetics [*Liebowitz, J.*, 1993]. A genetic algorithm maintains a list of possible solutions to a problem. Based on whether or not the previous solutions were successful, the fittest solutions not only survive but also exchange information with other candidates to form new solutions. Genetic algorithms are useful for inductive learning, conflict resolution, and classification. Some applications of genetics algorithms are scheduling systems and systems training neural networks. Genetics algorithms integrated with rule-based systems enable developing systems that can generate new rules.

All those new hybrid systems promise a new generation of applications but there are still many obstacles ahead researchers. One of the greatest challenges arising when integrating knowledge-based systems with other technologies, including conventional-computing technologies, are software integration problems. One solution is using common communication protocols such as DDE. This results in loosely integrated systems. Another approach is using object-oriented programming to glue different technologies what results in a tightly integrated systems. This approach was used in CADIE (see chapter 5). However, even using object-oriented techniques to blend different technologies does not resolve all the integration problems, because standards are just beginning to emerge in the object-oriented world [*Liebowitz, J.*, 1993].

On one hand, all those new tools and techniques will result in more powerful and easier to use applications to the benefit of end users. On the other hand, more skills and knowledge will be required from application developers. In the end, however, an application should dictate tools and techniques that should be used. It means that for simple applications there is no need of using those new tools and techniques, traditional techniques will work just as well or better. It also means that developers of complex applications will have in their arsenal a new set of tools and techniques to choose from.

References

Agarwal, R., et al., *Knowledge Base Maintenance, Expert Systems: Planning, Implementation, Integration,* (Summer 1991).

Beynon-Davies, P., Expert Database Systems, A Gentle Introduction, McGraw-Hill Book Company Ltd., England, 1991.

Tom Brooke, *The Art of Production Systems,*AI Expert, January 1992.

C. Forgy, *Rete: A Fast Algorithm for the Many-Pattern/Many-Object-Pattern Match Problem,* Artificial Intelligence, Vol. 19, No. 1, Sep. 1982, pp. 17-37.

David W. Franke, *Imbedding Rule Inferencing in Applications*, IEEE Expert, Dec. 1990.

Gardarin, G. and E. Gellenbe, *New applications of Database Systems,* Academic Press, London, 1984.

Harmon, P., R. Maus, and W. Morrissey, *Expert Systems Tools And Applications,*John Wiley and Sons, New York ,1988.

Harmon, P., and B. Sawyer, *Creating Expert Systems,* John Wiley & Sons, New York, 1990.

Hayes-Roth, F., et al., *Building Expert Systems,* Addison-Wesley, Reading, MA, 1983.

Hedberg, S., *New Knowledge Tools,* Byte Magazine, July 1993, p. 106-111.

Hedberg, S., *See, Hear, Learn,* Byte Magazine, July 1993, p. 119-128.

Van Horn, M. *Understanding Expert Systems,* Bantam Books, Toronto 1986.

H.C. Howard & D.R. Rehak, *KADBASE, Interfacing Expert Systems with Databases,* IEEE Expert, Fall 1989.

Jarke, M. and Y. Vassiliou, *Databases and expert systems: opportunities and architectures for integration,* In Gardarin and Gellenbe, 1984.

Kappa-PC Advanced Topics, Intellicorp, June 1992.

Kappa-PC User's Guide, Intellicorp, June 1992.

Level5 Object User's Guide, Information Builders, Inc., 1990

Level5 Object Reference Guide, Information Builders, Inc., 1990

Liebowitz, J., *Roll Your Own Hybrids,* Byte Magazine, July 1993, p. 113-115.

William Mettrey, *A Comparative Evaluation of Expert System Tools,*Computer, February 1991.

William Mettrey, *An Assessment of Tools for Building Large Knowledge-Based Systems,* AI Magazine, Vol.8, No. 4, Winter 1987, pp. 81-89.

Robert J. Mockler & D.G. Dologite, *An Introduction to Expert Systems,*Macmillan Publishing Company, New York, 1992.

Edmund C. Payne & Robert C. McArthur, *Developing Expert Systems,* John Wiley & Sons, Inc., U.S.A., 1990.

Prerau, D. S., *Developing and Managing Expert Systems,* Addison-Wesley, Reading, MA, 1990.

Quinlan, J. R., *Applications of Expert Systems (Vol. 1),* Addison-Wesley, Sydney, 1988.

Earl D. Sacerdoti, *The Copernican View of Artificial Intelligence,*Sun Technology, Winter 1989.

Schorr, Herbert, and Alain Rappaport, *Innovative Applications of Artificial Intelligence,* AAAI Press, Menlo Park, CA, 1989.

Thompson, T., *PowerPC Performs for Less,* Byte Magazine, August 1993, p. 56-74.

Efraim Turban, *Decision Support And Expert Systems: Management Support Systems,* Macmillan Publishing Company, New York 1993.

E. Turban and P. Watkins, *Applied Expert Systems,* Amsterdam, 1988.

Waterman, D. A., *A Guide to Expert Systems,* Addison-Wesley, MA, 1986.