An Expert-System Shell Using Structured Knowledge

An Object-Oriented Approach

K.S. Leung and M.H. Wong
Chinese University of Hong Kong

Most rule-based expert systems[1] are not flexible enough to handle problems that mix logical deductions, rule-based inferences, and procedure execution, but mixed problems are common in modern society. Management decisions, for example, are often based on both rules and operations-research techniques. These problems cannot be solved by either a conventional expert system or common operations-research techniques alone. Their solution demands a new system architecture that effectively combines rules and procedures.

Some conventional expert systems can acquire consultative data only through interactive input; thus, they are not suitable for problems involving large amounts of data. To overcome this drawback, automatic feeding of data from a database should be considered an essential feature of a new system architecture. Many present-day expert systems also take an unstructured approach to knowledge representation; thus, when the size of the knowledge base increases significantly, the knowledge can become unmanageable. To overcome this problem, future expert-system shells will need to use structured knowledge-representation methods.[2]

Two expert systems built with this prototype shell have demonstrated its power and flexibility. It mixes declarative and procedural knowledge, overcoming a major problem of conventional shells.

This article presents a new architecture for an expert-system shell. Its structured knowledge representations enable mixing rules with procedures. Its built-in database interface not only allows automatic extraction of data from a database management system, it also provides a fuzzy database query facility. In addition, the shell's object-oriented approach[3] to knowledge representation supports data and knowledge abstraction. The approach also encourages modular designs, which improve the efficiency of knowledge acquisition and management. Another feature is encapsulation, which prevents object manipulation except by defined operations. Overall, this object-oriented approach improves the consistency, maintainability, understandability, and modifiability of the knowledge base, and, because it minimizes object interdependency,[4] the knowledge can be structured.

Our prototype expert-system shell, called System X-I, includes a flexible knowledge-acquisition module, an inference engine (backward chaining), a database interface, fuzzy retrieval facilities,[5] and an external methods interface. The flexibility and power of System X-I have been proven in several implementations. Two case studies are presented later in the article, but first, let's compare representation methods and take a look at the shell's overall architecture and object-oriented approach.

0018-9162/90/0300-0038$01.00 © 1990 IEEE

Classical versus object-oriented knowledge representation

A knowledge-representation method is the way a knowledge engineer models the facts and relationships of the domain knowledge. Classical methods, such as semantic networks, object-attribute-value triplets, frames, and rules, are commonly used in classical expert systems. Each method has its own advantages and disadvantages.

Semantic networks. A semantic network[6] is composed of two tuples (N, L). N is a set of nodes used to represent objects and descriptors. An object may be a physical object or a conceptual entity, while a descriptor provides additional information about the object. L is a set of links connecting the nodes and representing the relations among them. Some commonly used links are is-a and has-a links. An is-a link represents class and instance relationships and usually has an inheritance feature. A has-a link shows that a node has a certain property. Semantic networks also have definitional links for representing declarative relations.

Object-attribute-value triplets. In the object-attribute-value method, knowledge is represented by three tuples (O, A, V). O is the set of objects, which may be physical or conceptual entities. A is the set of attributes, which are general characteristics or properties associated with objects. Values, V, specify the natures of the attributes.

The OAV method is actually a special form of semantic network. The relation between an object and an attribute is a has-a link, and the relation between an attribute and a value is an is-a link. The objects, attributes, and values of OAV are equivalent to the nodes in semantic networks. Knowledge can be divided into dynamic and static portions. The triplet values are the dynamic portion. These values may change, but the static portion (usually facts and rules) remains unchanged for different consultations.

OAV is more structured than a semantic network. However, when the number of objects increases, an OAV system becomes difficult to manage.

Frames. A frame is used to describe an object.[7] It is composed of slots storing information associated with the object. The function of the slots is similar to that of the attributes in OAV. However, frames differ from OAV in that the slots may contain default values, pointers to other frames, sets of rules, or procedures. Frames may also be linked to allow for inheritance.

Ultimately, frames and OAV can be considered special cases, or subsets, of semantic networks. The representational power of the three systems is the same. The difference lies in the structure and concept of their knowledge organizations.

Rules. A rule has two parts. The first part is a premise of conditions connected by logical-AND or logical-OR relationships. The second part is a conclusion. When the premise of a rule is true, the conclusion of the rule will become true. In some systems, rules may be implemented by semantic network or OAV, as in Mycin,[8] the medical diagnostic system developed at Stanford University. Alternatively, rules may be represented by frames, as in IntelliCorp's Knowledge Engineering Environment.[9]

Evaluation and comparison. The major advantage of semantic networks is flexibility, since new nodes and links can be defined as required without restriction. This flexibility also exists in object-oriented knowledge representations where, by storing the names of other objects as the attributes of an instance object, relations between instance objects can be established dynamically. These relationships have the same power as links in semantic networks; in fact, this object-oriented construct can be viewed as a dynamic semantic network. The is-a links of semantic networks can be implemented in object-oriented representations by relationships between classes and subclasses or between classes and instances. Has-a links can be implemented by the relationships between classes and attributes. Therefore, object-oriented knowledge representation has the same power as a semantic network but is much more structured.

A common disadvantage in semantic networks, rules, and OAV representations is that they are not structured enough. A significant increase in the number of objects or rules makes the system difficult to manage. This is because the knowledge cannot be modularized and interactions among rules and objects become too complex. When the value of an object or an attribute is modified, it is difficult to pinpoint the effects on the whole system. Therefore, such knowledge representations are difficult to develop and maintain, especially for a large knowledge base. The encapsulation property and structuredness of object-oriented knowledge representations give them a distinct edge over these three representations.

Frames are more structured than semantic networks, rules, and OAV representations, since related attributes and rules can be grouped into frames hierarchically. However, modularity of knowledge represented in frames cannot be clearly defined, and frame representation lacks flexibility. In a frame system, relationships between frames may be member or subclass links and thus are not unique. Moreover, in some systems, a rule is represented by a frame linked to another frame with a special relationship. These factors greatly reduce the structure in a frame system. In object-oriented knowledge representation, which is quite similar to frames, knowledge can be arranged in a hierarchical form using classes. However, a subclass link is the only possible relationship between two classes, an is-a link is the only possible relationship between a class and an instance object, and rules are defined as methods in classes — clear-cut distinctions that reduce ambiguity and improve understandability.

System X-I architecture

Our expert-system shell, System X-I, has four functional components: an inference engine, an end-user interface, a knowledge-acquisition module, and an environment for domain knowledge. System X-I adopts an object-oriented approach for knowledge representations and inferences, a relatively new approach in expert-system shells. The knowledge representations in System X-I include both declarative and procedural forms. External routines written in other languages can also be invoked as procedural knowledge. Figure 1 shows the system's overall architecture.

The expert-system shell contains many objects. All external interfaces and inference and control mechanisms are implemented by system methods in system objects. The inference engine is used to support the system methods for rule inference. All system objects are grouped in the system class "root." Knowledge engineers can use this class without any declaration.

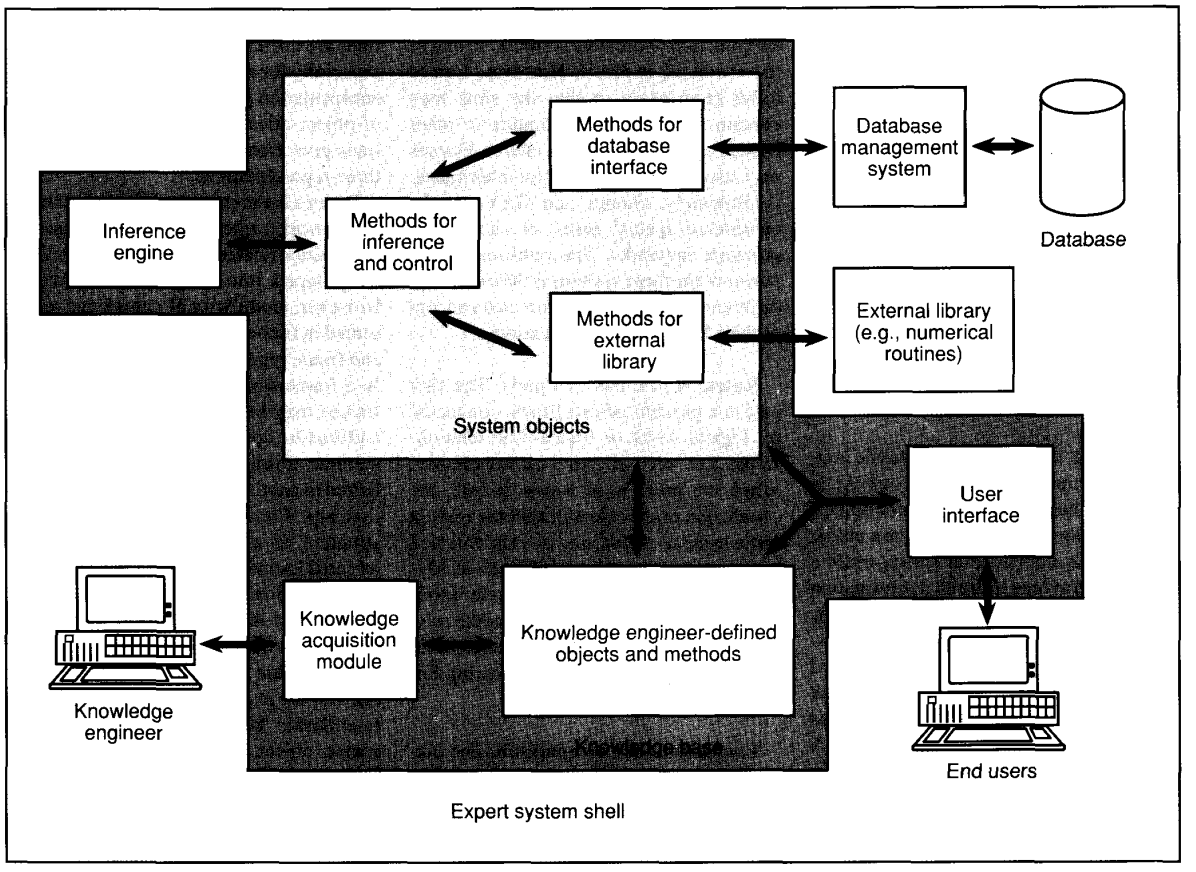

Figure 1. Architecture of System X-I.

A knowledge base of an expert system built from System X-I is also defined by a collection of classes, objects, and methods. If a user-defined class is declared a child of the root class, it can inherit all the capabilities of the system methods. Thus, each user-defined object may be treated as an independent entity that inherits inference capabilities from the system class "root."

End-users can communicate with any object defined by the system or by knowledge engineers through the user interface. The user interface translates user input into messages for invoking appropriate objects in the system. Messages sent to the user interface are decoded, and the resulting information is output to the screen. The knowledge-acquisition module captures the domain knowledge supplied by knowledge engineers, transforms it into internal format, and stores it in a knowledge base.

To meet the needs of different users, System X-I supports two modes of knowledge input. Knowledge engineers familiar with the system and with knowledge representation can input knowledge quickly through conventional editors. The second mode, for knowledge engineers or domain experts who do not have a clear concept of the knowledge representation method, supports input through interrogation.

In interrogation mode, the knowledge engineer selects an option from a menu that includes inserting classes, inserting instance objects, modifying classes, and modifying objects. For the option of creating a class, the knowledge engineer must input some compulsory information, such as the name and parent of the class. An instance object can be created in a similar manner. An option for inputting optional information, such as class and instance variables and attributes, can then be selected. Rules are input as an optional type of attribute. To modify a class or an instance object, the knowledge engineer chooses the appropriate class or instance object from a menu. Then, a series of questions or menus are used to identify and modify the piece of knowledge.

Objects

System X-I represents knowledge by a collection of objects. An object is an independent entity represented by some data and a set of operations (methods and capabilities).[10] Therefore, an object can be used to represent a variety of knowledge. Knowledge (K) can be formally represented by three tuples, $K = (C, I, A)$, where C is a set of classes and I is a set of instances represented by class and instance objects, respectively. A is a set of attributes possessed by the classes and instances. The behaviors of C, I, and A are restricted by the law predefined in the shell. The law

includes rules for inheritance, message passing, and encapsulation.

Classes. A class is a description of a group of similar instance objects.[10] It is a mold that determines the behavior of its instances. Each class has a unique name and a set of attributes that define the properties of the class. A class may be a subclass of another class and may inherit properties from its parent class (see "Inheritance" subsection for details).

Five sets of attributes may exist in a class object:

(1) Name — the name of the class; used to reference a class in the system.

(2) Super_class — the name of the parent of the class.

(3) Class_variables — a set of variables shared among all instances of the class, that is, global variables accessible by all instances of the class. Get and Store operations may be performed on them.

(4) Instance variables — the set of variables possessed by each instance of the class. Variables may be classified into data and database data. Each instance may have different values for these variables.

(5) Class attributes — the set of methods, external methods, and rules shared by all instances of the class.

Instance objects. Instance objects are members of classes. Their properties are defined by their parent classes. Each instance object consists of three sets of attributes:

(1) Name — the name of the object, which is unique in the system. It is used to identify the object.

(2) Class — the class that contains the object.

(3) Instance attributes — attributes belonging to the instance object. Some operations may be performed on these attributes. The behavior of the object is determined by the values of these attributes.

In general, an instance object i is defined by its class and the values of its attributes: $i \in I$, where I is the set of instances in the system; $\forall\, i \in I$, attribute x with any value v is valid for i, if val(x) = v and x are defined in class(i), where class(i) is a function that returns the name of the class of instance i and val(x) is a function that returns the value for attribute x.

Attributes. The property of an attribute is determined by its type and value. The type of an attribute is defined by its class, while the value may be defined in its class or its instance object. In System X-I, attributes may be classified into three types:

(1) Data — Attributes of this type are used to store simple data types such as integer, boolean, real, and string. Get and Store operations may be applied to these attributes.

(2) Database data — An attribute of this type corresponds to one value of a field in a relational database of VAX Rdb. To define this type of attribute, the knowledge engineer must specify the relation name, field name, field type, and the key field defined in the database. The name of the attribute may be different from that of the field defined in the database.

When a method is invoked to get the value of an attribute belonging to this type, the value of the data will be drawn automatically from the database through a database interface.[5] Besides drawing data, the interface provides system methods to draw information with relational operations and to handle fuzzy queries, such as "select a senior man who is at least about 5 feet 8 inches tall."

(3) Methods — This type of attribute is not for storing data but for defining capabilities. (See section titled "Methods" for details.)

Inheritance. Properties of a class can be inherited from its parent's class. This feature permits factoring knowledge into a class hierarchy. Thus, it encourages modular design of knowledge. The system adopts the inheritance rules, which are similar to those in Smalltalk,[10] as follows:

(1) If class A inherits from class B, then the objects of class A support all operations supported by objects of class B.

(2) If class A inherits from class B, then class A's attributes are a superset of class B's attributes.

Therefore, $\forall\, c \in C$ (the set of all classes), attribute x is valid for c if either $x \in$ class_attributes (c) or $x \in$ class_attributes (superclasses of c), where class_attributes (c) = the set of attributes of c.

Methods

Methods are a kind of attribute belonging to objects. They are used to represent capabilities, not to store data, and are defined in classes. Methods cannot be modified during consultation. In System X-I, methods are further classified into three types: rules, internal methods, and external methods.

Rules. In System X-I, rules are methods belonging to objects. They are used to find the values of the variable attributes. The corresponding rules are triggered by the system method Find-Val if the value of a variable attribute is undefined in its instance object or database. If the condition of a rule is satisfied, the conclusion will determine the value of the corresponding attribute.

The format of a rule can be expressed as follows:

Rule :== if *Condition* then *Conclusion*

The form of *Conclusion* is

Conclusion :== <attributes> is <value>

The <value> may be a message or a datum of a primitive type, such as integer, real, string, or list. If <value> is a message, the method specified in the message will be invoked, and the result will be assigned to the attribute. Such a design allows a rule to trigger a method, thus achieving a free intermixing of rules and procedural knowledge. In addition, rules can be chained across objects within a class or an encapsulated module. If <value> is a datum, the value will be directly assigned to the attribute.

The structure of *Condition* may be expressed in extended Backus-Naur form as follows:

Condition :==
(AND *Condition Condition**) |
(OR *Condition Condition**) |
(*Operator* <attribute> <value> |
(*Unknown* <attribute>)
Operator :==
>|<| = |>=|<=

The value of a condition is either true or false. A condition may be composed of a number of conditions connected by logical-ANDs or logical-ORs. Therefore, a condition can be defined recursively as shown above. The basic terms of a condition are (*Operator* <attributes> <value>) and (*Unknown* <attributes>). In addition, fuzzy concepts can be handled by rules, as reported in our earlier article on System Z-II.[11]

Internal methods. Internal methods are used to define knowledge in a procedural

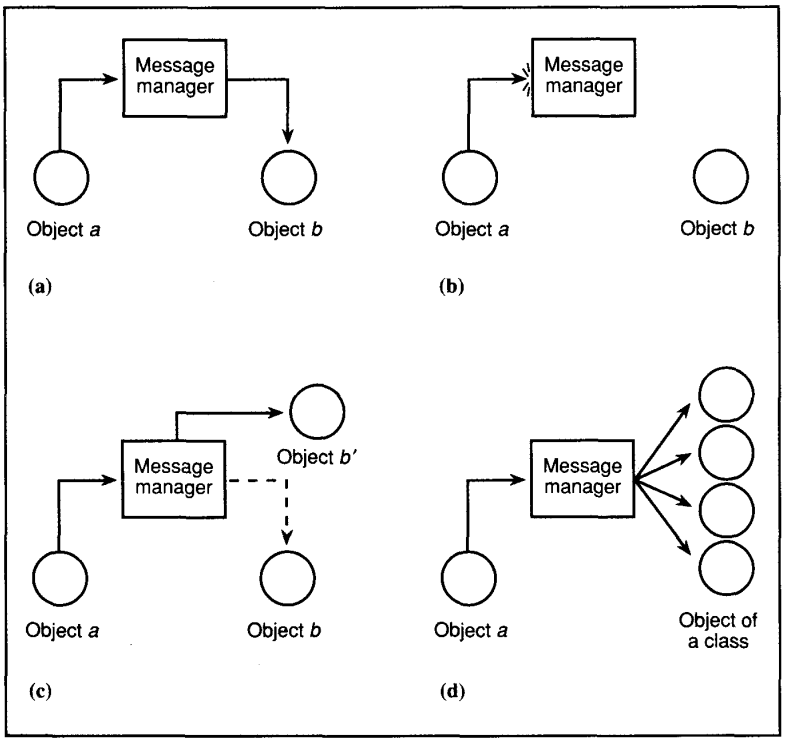

Figure 2. Four possible actions by the message manager: (a) direct delivery, (b) block, (c) reroute, and (d) broadcast.

form. The format of an internal method is

(<attribute> method (lambda <argument list> <method body>))

The <attribute> is the name of the method. The <argument list> is a list of the parameters needed by the method. The <method body> is a series of statements that are method steps. The syntax of the procedure body conforms to Common Lisp. Since messages can be sent by a method, a method can trigger another method or rules.

External methods. External methods are system-supplied or user-defined procedures written in any computer language. They can be viewed as tools used by the objects. For example, they may be routines for the Simplex method, linear programming, integer programming, and so forth. Since the expert-system shell is developed on VAX computers using a VMS operating system, these procedures must conform to VAX procedure-calling and condition-handling standards; otherwise, they cannot be linked with the expert-system shell. A knowledge engineer only has to define an interface for a procedure. Once the interfaces are defined, external methods can be used as if they were internal methods. The format of the interfaces is defined in extended Backus-Naur form as follows:

External_method :==
  (<attribute> external_method
  (:file <filename>
  [:entry-point <procedure name>]
  [:result <result type>])
  *Argument**
Argument :==
  (<argument>
    [:access *Access_method*]
    :lisp-type <Ltype>
    [:vax-type <Vtype>])
Access_method :==
  :in | :out

The <attribute> is the external method's name for internal identification. The <filename> is the name of the file in which the external procedure is stored. The <procedure name> is the name of the procedure and is optional. If it is neglected, the procedure name is assumed to be the same as the method name (<attribute>). The <result type> is the data type of the value returned from the procedure. If no value is to be returned, this field may be omitted. <Ltype> and <Vtype> are the types of the arguments interpreted by Lisp and the VAX/VMS system, respectively.

Controls and message passing

System controls can be classified as intra-object and inter-object controls. Intra-object control is regarded as object self-coordination governed by the object's capabilities (methods). In other words, intra-object control depends on the domain knowledge input by a knowledge engineer, and it determines the system's microbehavior. Inter-object control, on the other hand, coordinates interactions among objects through inter-object communication. Message passing is the only means of communication between objects. The system's macrobehavior is determined by the message-passing mechanism. Since message passing is the only way to activate an object, it enhances knowledge modularity and encapsulation.

Message format. Messages activate the execution of methods in objects. A message consists of four components as follows:

(1) Sender — the name of the object sending the message. In System X-I, the sender need not be specified explicitly in the message.

(2) Receiver — an item specifying the objects to which the message is sent. The item usually contains the name of an object (receiver). However, sometimes the name of the receiver is not known or cannot be specified directly, and, of course, messages are sometimes broadcast to multiple receivers. System X-I has facilities to handle such cases. First, the name of a receiver may be specified indirectly by storing it in an attribute of the sending object. Second, a message may be sent to the sender itself by leaving the receiver field as an empty list. Third, a message may be sent to a list of receivers. Finally, a message may be sent (broadcast) to each instance of a class.

(3) Selector — an item specifying which method the message is to invoke.

Usually it is the name of the method.

(4) Arguments — parameters to be passed to the method.

Message management. After creation by an object and before transmission to the receiver, a message is sent to the message manager for examination. If the message is valid and requires no special handling, the message manager redirects it to the receiver. In other words, all exchanges are subject to system law,[12] but the process is, of course, transparent to users.

In general, the message manager has four possible options for each message. First, the original message can be delivered directly to the receiver specified by the sender. Second, the content and receiver of the message can be modified. Modification of the receiver reroutes the message to another object. For example, when the debug option of the system is set, a message originally sent to, say, object *A* may be rerouted to object *A*2, which contains methods to dump some system information. Third, the message can be blocked and not delivered to any object. This action may be taken because the message is invalid (the receiver does not exist or the required method cannot be found) or because the sender does not have sufficient privilege to invoke a particular method in the receiver. Finally, the message can be broadcast to all instances of a class. These four situations are shown in Figure 2a-d.

Encapsulation and modularity. Although classification hierarchies in frame systems seem very structured, interactions may still exist among instance objects (leaf nodes). Hence, frame systems cannot achieve the modularity of System X-I's object-oriented approach, which promotes encapsulation and modularity.

Encapsulation is a technique that reduces interdependence among different modules by defining strict external interfaces for each module.[4] A module of knowledge is encapsulated if access by other modules is restricted to predefined interfaces. Therefore, encapsulation is tied to the message management mechanisms. As stated above, messages can be blocked if the sender's privilege is insufficient to invoke certain receiver methods. This mechanism ensures encapsulation by preventing some primitive methods, such as Get and Store, from being invoked by other objects. An instance object's attributes are accessible only to the object itself. Therefore, when the object is rewritten or its attributes renamed, the effects on other objects are minimal, and encapsulation is achieved.

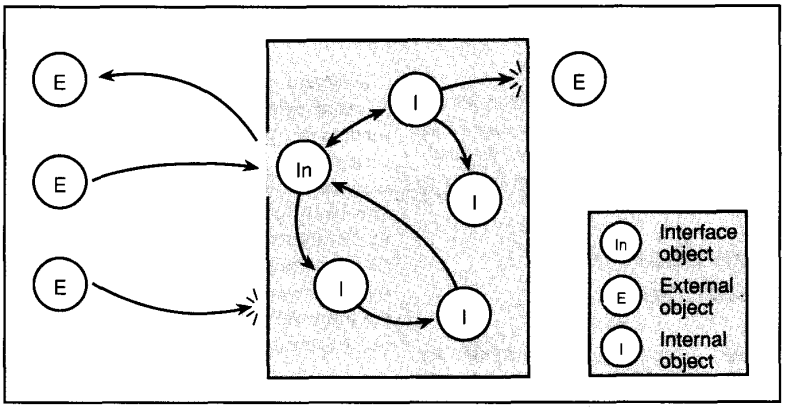

Figure 3. Encapsulation of a knowledge module containing several objects. Only interface objects can exchange messages with external objects.

The effects of encapsulation are closely related to modularity. With encapsulation, the contents of objects are accessible only via specified methods. For modularity, knowledge is separated into different modules. Since modules can interact only through predefined interfaces, they can be tested and modified separately. Knowledge organized in that form is quite structured and is more easily managed and understood.

A module of knowledge can be an object or a group of objects. If a module is an instance object, the message management mechanism has already guaranteed access only through predefined methods. To encapsulate a module composed of a group of objects, the module must be arranged so that no object in the module, except those responsible for interface, will exchange messages with objects outside the module. Figure 3 illustrates this group encapsulation, a unique feature of System X-I.

Inference tracing by time stamps

In System X-I consultations, rule and frame deductions and procedure executions can be freely mixed. For tracing and explanatory purposes, events are tracked using a time-stamp approach. Each message generated in the system is assigned a unique time stamp based on system time. In fact, system time is a counter of the number of messages received by the message manager. When the message result is returned to the sender, the return time is also marked. A unique time stamp, based on arrival and return times, is created and sent to the sender along with the value and certainty of the result. Therefore, each returned value stored in an instance variable has a certainty factor and a time stamp associated with it.

For example, if the arrival time is 5 and the return time is 10, a time stamp "S-5-10" will be created. As it creates the time stamp, the message manager also creates a record of all information about the message. This record can be retrieved with the time stamp as the key. The record includes the time stamp, receiver, selector, arguments, result, and an information list explaining how the result was drawn. It may also include the time stamps of the subgoals. Thus, an inference process can be traced.

In short, a time stamp has two functions: tracing the order of message activation and tracing the inference process.

(1) The activation order of messages can be traced by their time stamps. For example, if two messages have time stamps *S-a-b* and *S-c-d* and *c* and *d* have values between those of *a* and *b*, it can be deduced that message *S-a-b* directly or indirectly caused the generation of message *S-c-d*. In other words, the goal of finding the result of the message *S-c-d* is a subgoal of finding the result of message *S-a-b*.

(2) With time stamps, details of corresponding messages can be found. Thus, the process by which the system determined a

particular result can be traced and an explanation generated.

A sequential system is assumed for the above approach. Parallel operations using message passing require a more sophisticated time-stamping technique.

Classification of problems

Rules, procedures, and databases are frequently used independently to handle various classic computing and artificial intelligence problems, for instance, rules for rule-based expert systems, procedures for operations research problems, and databases for management information systems. However, in the X-I expert-system shell, all three techniques are represented as attributes of objects and can be intermixed in any proportion to model and solve complex problems. Rules and procedures usually drive and control the problem-solving process, while databases play the supportive role of automatic data management. The top-level control of an expert system built using System X-I is driven by either rules (goal directed) or procedures, although they can be freely mixed at lower levels. With System X-I, a knowledge engineer can use either, modeling the problem according to his or her own style of working.

Goal-directed problems. This class of problems uses rules for top-level control of the expert system. In System X-I, these systems are similar to conventional goal-directed (rule-based) systems in which the goal can be preset as a system goal or set according to user queries during consultation. The system method "Find-val" is triggered when a message is sent to an instance object to find the value of an appropriate attribute corresponding to a query. The values of attributes may be obtained from objects or a database or through rule inferences. In a rule inference process, messages may be generated and sent to other objects for finding intermediate values to support the inference.

Procedural-driven problems. This class of problems uses procedures for top-level control. However, such an expert system built using System X-I can freely use heuristics, databases, and other procedures at lower levels to support the problem-solving process.

Case studies

Different problems require different mixes of techniques and knowledge representations. Among the existing expert-system shells, few possess all the features required to solve all types of problems effectively. In this section, we demonstrate the flexibility and power of System X-I in two case studies of expert systems belonging to different classes. In the first system, the top-level control is procedure-driven; in the second, it is goal-directed.

Case I: Distribution of courses to teachers. In our first case study, the problem is to assign computer courses to teachers. The information on each teacher is stored in a database. Each teacher's ability is measured in six dimensions: hardware, software, languages, business, theoretical, and numerical. Each course has a requirement, a difficulty index, and a time slot. The objective is to distribute these courses to suitable teachers such that no teacher is overloaded and no two courses with the same time are assigned to a teacher.

A procedure is used for the highest level of control. To solve the problem, it is modeled by three classes of objects: courses, staff, and main (main object for top-level control).

Class "courses." Each instance object of class "courses" represents a course and possesses the following data and capabilities (methods).

Data:

(1) Name — the name of the instance object.

(2) Class — the class to which the object belongs. In this case, it will be courses.

(3) Difficulty index — a scalar quantity indicating course difficulty and workload.

(4) Time — the time for the course.

(5) Requirements — the required background of the assigned teacher. The requirements can be given in a restricted natural-language form of a fuzzy query language.[5]

(6) Id — the identity number of the teacher to whom the course is currently assigned.

(7) Suitability — the degree of membership,[11] a value between 0 and 1 reflecting the assigned teacher's ability in teaching the subject.

Methods:

(1) Select-teacher — a procedure used to select a list of teachers who satisfy the requirements, specified as the data of the course object in the fuzzy query language supported by the database interface.[5] It will invoke the "assign" method to select a teacher.

(2) Assign — a procedure to select a teacher from the list obtained by the above method. A message will then be sent to the teacher object to request him or her to teach the course. If the teacher object refuses the course, another teacher will be selected.

(3) Exchange — method to check whether the suitability of the assigned teacher is below a certain value. After the initial assignments, the main object will broadcast a message to all course objects to invoke the exchange method. Each course object will be checked by the method, which uses rules to determine a teacher's suitability for a course. For example, if the suitability is less than 0.5, an exchange will be initiated.

(4) Accept-exchange — a method that uses rules to determine whether an exchange of courses between two teachers is acceptable. This method is invoked by the above exchange method.

(5) List-assign — a procedure to print the identity number of a teacher assigned to a course, together with his or her suitability for teaching the course.

Class "staff." Each instance object of class "staff" represents a teacher. These teacher objects possess four data (besides a name and class slots) and one method as follows.

Data:

(1) Limit — the maximum workload that can be assigned to the teacher.

(2) Workload — the current workload assigned to the teacher.

(3) Subject-list — a list of courses assigned to the teacher.

(4) Time-list — a list of times of the courses assigned to the teacher.

Method:

(1) Allocate — a method invoked by the assign method of a course object. The teacher object will accept the course allocation if there is no conflict in the teacher's timetable and the teacher is not overloaded.

Class "main." This class has only one instance object, which is the main object responsible for the top-level control of this course-allocation expert system. This top-level control object contains three steps in procedural form. Step one broadcasts a message to all courses objects invoking

them to find teachers for themselves. Step two broadcasts a message to all courses objects asking them to exchange teachers if theirs are not suitable. Step three broadcasts a message to all courses objects requesting a printout of the teachers assigned.

This expert system illustrates a natural problem-solving methodology modeled by classes, objects, methods, and data. The relations between the courses and teachers are not explicitly declared in classes, but they are implicitly linked by methods and message passing. Such links are active, dynamic, and more flexible than the links in semantic networks or frames.

Since this is an NP-hard problem, an optimal solution is very time consuming. This expert system solves the problem in much the same way a person would. Although the solution is not optimal, it is acceptable. (For the results of this case study, see the sidebar at right.)

Case II: Management-decision expert system. System X-I can also develop an expert system that supports decision-making. In this case study of a goal-directed management problem, top-level control is rule-based, but the lower levels of problem solving involve both procedures and rules.

The Pennwood Corporation has designed a new air conditioning system. The product will sell to contractors for $1,400, and the cost to the home buyer will be about $2,600. The estimated manufacturing cost is either $900 or $1,050. The exact cost cannot be determined because of pending labor negotiations in the material supplier's industry. If there is a substantial wage increase or a strike, the cost will be $1,050. If the negotiations are successful and workers agree to a smaller wage increase, the cost will be $900.

However, if Pennwood decides to manufacture and market the product, the project will require a $400,000 investment, including sales promotion and facility expansion. For the project to be successful, sales must cover this investment and allow for profit. Based on a preliminary analysis, sales are determined by two factors: economic conditions affecting new home construction and the competitive situation within the air conditioning industry.

Because of the air conditioning system's $2,600 price, Pennwood's Marketing Division sees a sales potential only for houses priced at or over $50,000. Preliminary market research indicates that buyers of approximately three percent of all new homes in this price range would install the air conditioning system if it were available for $2,600. Market research estimates construction of 60,000 residences in this price range under good economic conditions, but only 40,000 if conditions are bad.

Another marketing factor is the presence or absence of competition. If a competitor brings a similar model to market, it is expected that the firms will share the market equally. If not, Pennwood could expect to get the entire market.

The objective is to determine whether the investment should be made. The expert system is modeled by two classes of objects, "products" and "air-cond," both of which possess only one instance object.

Case I results

In this demonstration case, 10 courses are to be assigned to five teachers. A typical course object is shown as

Name: CSC282
Class: Courses (meaning object CSC282 belongs to class "courses")
Difficulty: 90 (an index given out of 100)
Time: M1 (Monday, first period)
Requirements: The teacher should have a strong background in both software and numerical analysis.

The requirement is expressed in a fuzzy query condition, which is used by the database interface to select suitable teachers.[5]

Each teacher is modeled by an object with attributes name, class, current workload, and upper limit of workload. Three teacher objects belong to the class "staff" and two belong to "senior-staff," a sub-class of staff. The initial workloads for all teachers are zero. The workload upper limits are 250 for staff and 100 for senior-staff.

Five teacher objects with initial data are given as follows:

(staff1 (class staff)(workload 0)(limit 250))
(staff2 (class staff)(workload 0)(limit 250))
(staff3 (class staff)(workload 0)(limit 250))
(staff4 (class senior-staff)(workload 0)(limit 100))
(staff5 (class senior-staff)(workload 0)(limit 100))

The database contains the name, identity number (Id), and abilities of each teacher. The ability of a teacher in each of the six areas, namely, hardware (Hard), software (Soft), language (Lang), business (Bus), theoretical (Theo), and numerical (Num), is represented by a scalar quantity ranging from 0 to 10 as follows:

Name	Id	Hard	Soft	Lang	Bus	Theo	Num
Peter	1	7	9	9	6	9	8
David	2	9	9	8	9	7	6
John	3	8	7	6	8	8	5
Mary	4	7	8	8	9	9	6
Betty	5	8	7	6	8	9	6

The expert system gave the following results after initial allocations and execution of the method "exchange."

Teacher Id	Courses	Workloads
1	CSC150E, CSC150D, CSC282	228
2	CSC170, CSC150C, CSC150B	212
3	CSC212, CSC142	185
4	CSC150A	80
5	CSC414	75

Case II results

Three consultations for the management problem presented in Case II yielded the following results.

Consultation 1

What is the value of CONTRACT-PRICE in object AIR-CONDITION-SYSTEM
input> 1,400
What is the value of NEGOTIATION in object AIR-CONDITION-SYSTEM
input> success
What is the value of NO-OF-HOUSE-OWNER in object AIR-CONDITION-SYSTEM
input> 60,000
What is the value of COMPETITOR-NO in object AIR-CONDITION-SYSTEM
input> 0
What is the value of MARKET-PRICE in object AIR-CONDITION-SYSTEM
input> 2,600
What is the value of FIX-COST in object AIR-CONDITION-SYSTEM
input> 400,000
Conclusion:
The investment is strongly recommended
Demand = 3%
Sales = 1800 units
Profit = $500000

Consultation 2

What is the value of CONTRACT-PRICE in object AIR-CONDITION-SYSTEM
input> 1,400
What is the value of NEGOTIATION in object AIR-CONDITION-SYSTEM
input> fail
What is the value of NO-OF-HOUSE-OWNER in object AIR-CONDITION-SYSTEM
input> 40,000
What is the value of COMPETITOR-NO in object AIR-CONDITION-SYSTEM
input> 2
What is the value of MARKET-PRICE in object AIR-CONDITION-SYSTEM
input> 2,600
What is the value of FIX-COST in object AIR-CONDITION-SYSTEM
input> 400,000
Conclusion:
Do not invest in the project
Demand = 3%
Sales = 400 units
Profit = – $260,000 (loss)

Consultation 3

What is the value of CONTRACT-PRICE in object AIR-CONDITION-SYSTEM
input> 1,200
What is the value of NEGOTIATION in object AIR-CONDITION-SYSTEM
input> success
What is the value of NO-OF-HOUSE-OWNER in object AIR-CONDITION-SYSTEM
input> 45,000
What is the value of COMPETITOR-NO in object AIR-CONDITION-SYSTEM
input> 0
What is the value of MARKET-PRICE in object AIR-CONDITION-SYSTEM
input> 2,400
What is the value of FIX-COST in object AIR-CONDITION-SYSTEM
input> 600,000
Conclusion:
It is profitable but there may be some risk
Demand = 5%
Sales = 22,500 units
Profit = $75,000

Class "products." Class "products" represents general knowledge that may be applied to any company product. This class contains one datum and two methods.

Datum:

(1) Fix-cost — a datum used to store the fixed cost of the product. The value will be requested interactively during consultation.

Methods:

(1) Recommendation — an attribute related to the conclusion drawn, based on a set of rules to determine profitability. This attribute is, in fact, the goal of the rule-driven expert system. The recommendation will be given to the users at the end of the consultation. The rules are

If profit > fix-cost
Then investment is strongly approved

If profit > 0 and profit < fix-cost
Then investment is approved with warning

If profit < 0
Then investment is disapproved

(2) Profit — a method used to find product profit. The formula is profit = (contract price – variable cost) * sales – fix-cost

Class "air-cond." This class represents knowledge specified for the air conditioning system only. Because "air-cond" is a subclass of "products," it inherits all the knowledge of the products class. A knowledge engineer may input the data values to the instance objects before consultation. If the data values are undefined, the user will be asked to input the values during consultation. Class air-cond has the following data and methods.

Data:

(1) Contract price — the price of the air conditioning system sold to contractors.

(2) No-of-house-owner — the estimated number of new home owners in the coming year.

(3) Competitor-no — the estimated number of competitors in the coming year.

(4) Market price — the price of the air conditioning system sold to home buyers.

Methods:

(1) Variable-cost — determined by the result of the labor negotiation. Rules relate the variable-cost to the negotiation result.

Advance Announcement

The First International Conference on Systems Integration

April 23-26, 1990 Headquarters Plaza Hotel, Morristown, New Jersey, USA

Sponsored by
Institute for Integrated Systems Research
Department of Computer & Information Science
New Jersey Institute of Technology (NJIT)

In cooperation with
IEEE Computer Society
ACM (Association for Computing Machinery)
AT&T
Bell Communications Research (Bellcore)
Gesellschaft Fuer Mathematik und Datenverarbeitung(GMD)

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

ASSOCIATION FOR COMPUTING MACHINERY

This conference focuses on the problems, issues and solutions of integrated system design, implementation and performance. Integration technologies will be emphasized with a focus on computer-aided software engineering, collaborative and distributed systems, and computer integrated manufacturing systems. The conference will provide an international and interdisciplinary forum in which researchers and practitioners can share novel research, engineering development and strategic management experiences.

Tutorial-in-Brief

Monday, April 23, 1990 8:30 am - 5:00 pm

Tutorial 1: Interconnection Networks for Computer Systems Integration
Chuan-Lin Wu, University of Texas at Austin
Tutorial 2: Software Engineering with Abstractions
Valdis Berzins, Naval Postgraduate School
Tutorial 3A: Optical Data and Knowledge Base Machines
P. Bruce Berra, Syracuse University
Tutorial 3B: The Object-Oriented Paradigm and Integrated Systems
Erich J. Neuhold, IPSI-GMD
Tutorial 4: Systems Engineering Management: Unifying Principles for Systems Development Integration
Tom Gilb, Independent Consultant
Tutorial 5: The Future of Machine Vision in Manufacturing
Robert Mitchell, University of Texas at Arlington

Conference-in-Brief

Plenary Session Speakers
- *Arno A. Penzias,* AT&T Bell Laboratories
  Tuesday, April 24, 1990 9:00 am - 10:30 am
- *Elaine R. Bond,* Chase Manhanttan Bank, N.A.
  Wednesday, April 25, 1990 9:00 am - 10:00 am
- *Raymond T. Yeh,* Syscorp International, Inc.
  Thursday, April 26, 1990 9:00 am - 10:00 am

Dinner Banquet Speaker
Tom Gilb, Independent Consultant
Wednesday, April 25, 1990 7:00 pm - 9:00 pm

The Conference will be held at the Headquarters Plaza Hotel, 3 Headquarter Plaza, Morristown, New Jersey, USA. For reservation call (201) 898-9100 or (800) 225-1942. Please identify yourself as Systems Integration Conference attendant to receive the room rate for single occupancy $120/night or double occupancy $140/night. Reservations must be received by April 9, 1990.

Technical program

There will be three days for the presentation of 85 papers and four panels in three tracks. Papers and panels related to the following topics will be presented and discussed.
- System Integration
- Computer Architecture
- Artificial Intelligence
- System Evolutions
- Communications-Networking
- Panel: Integrated Design Manufacturing
- System Environments
- Communications-Protocols
- Objects Recognition
- Panel: Delivering Quality Systems
- Networking
- CAD/CAM
- Formal Specifications
- Data Management
- Manufacturing - Factory Floor Applications
- Standards and Definitions
- Data Management - Applied
- Manufacturing - Flexible Manufacturing Systems
- Panel: System Integration Leading to Distributed Systems
- Man-Machine Interfaces
- VLSI - Integration
- System Application
- Information Networking
- Distributed Systems
- Testing
- Information Systems
- Resource Management
- Design and Networking Issues
- Panel: Concluding Remarks

For a full Systems Integration Advance Program, please contact:
Professor Peter A. Ng
Institute for Integrated Systems Research
Dept. of Computer and Information Science
New Jersey Institute of Technology
Newark, NJ 07102, USA
Tel: (201)596-3387 Fax: (201)596-5777
E-mail: ng_p@vienna.njit.edu