Supporting Knowledge Migration in Organizations

Lawrence J. Williams
Frederick H. Lochovsky

July 25, 1998

Abstract

An organization typically possesses a large body of knowledge concerning its activities, as well as a set of policies governing the actions of its workers. Each individual in the organization also develops a body of knowledge concerning their own activities. In this setting, knowledge migration refers to the process by which an individual's knowledge *migrates* to become part of the organization's knowledge, and the policies defined by the organization *constrain* the actions of the individual. Generalizing from organizations to groups of individuals we illustrate how a planner can be used to combine the knowledge accessible to an individual into a candidate plan for solving the current problem. If this candidate plan satisfies all the applicable constraints, then it can be executed by the system.This approach utilizes multiple knowledge bases, each corresponding to an individual or a group, simple planning and constraint testing techniques, and knowledge base modification mechanisms to provide an environment that supports knowledge migration. We also discuss an implementation, within an object-oriented programming environment, of our approach to providing knowledge migration facilities.

1 INTRODUCTION

In organizations, knowledge about office work (tasks) resides both with the organization, usually in the form of goals to be achieved and/or outputs required, and with individuals in the form of specific mechanisms for achieving those goals/outputs [8]. In general, the organizational knowledge imposes constraints (policies) which the specific actions of the office workers performing a task must satisfy. In this setting, *knowledge migration* refers to the process by which an individual's methods for achieving certain goals/outputs become, over time, the organization's standard, and conversely, the policies defined by the organization constrain the actions of the individual.

In order to support knowledge migration, several subproblems must be addressed [4, 7, 16]. The system must be able to store the individual's problem solving knowledge and the organization's policies and problem solving knowledge in such a way that they can be modified, thus allowing the knowledge bases to grow [10]. The modification of a knowledge base may introduce inconsistencies which must be detected and resolved [9, 11]. Finally, the individual's and organization's knowledge, as well as the organization's policies, must be combined during the problem solving activities of an individual [5, 12, 18].

In section 2 we discuss our approach to the problem of supporting knowledge migration in organizations. Section 3 describes a prototype implementation, in Smalltalk, of this approach. Section 4 concludes the paper.

2 KNOWLEDGE MIGRATION

In our model of knowledge in an office, there is a large body of organization knowledge that defines both explicit ways of solving problems and more general constraints on the solution to a problem. There are also individual bodies of knowledge corresponding to the problem solving methods of each worker in the organization. Thus knowledge migration is the problem of *migrating* problem solving knowledge from an individual's knowledge base (IKB) to the organization's knowledge base (OKB), *incorporating* the organization's knowledge during the problem solving activities of the individuals, and *enforcing* the organization's constraints during these activities.

This structure can be generalized by allowing group knowledge bases (GKB), each performing the same role as the OKB, but only applicable to a subset of the office workers in the organization. This gives us the ability to support group-level knowledge and policies, utilizing this knowledge and enforcing these constraints only on the members of the group. A good example of this is an organization that is composed of a set of departments, with some departments wishing to enforce certain constraints upon their members above and beyond those specified by the organization. Similarly, this structure can be used to allow the easy addition of specialized knowledge that is to be used by only a subset of the office workers.

In this section, we define the structure and capabilities of the OKB, GKBs and IKBs, as well as the knowledge base modification and knowledge migration strategies. The conceptual system architecture is shown in Figure 1. An office worker will give the system a goal, representing a problem to be solved, or a request, representing a desired modification to the knowledge base. The system will then either use the planning and constraint testing mechanisms in an attempt to achieve the desired goal, or use the knowledge base modification facilities to modify a rule base.

2.1 Shared Knowledge Bases

From the definition of a GKB, it is apparent that the OKB is a GKB that applies to all the members of the organization. The structure and abilities of each are identical, and thus we will only describe the OKB in detail. An OKB has a set of domain rules (DR) which encode organization-defined problem solving knowledge, meta-level knowledge for use when modifying the DR, a set of constraint rules (CR) which define the policies that the organization has placed on the actions of the office workers performing a task, and meta-level knowledge that is used to modify the CR.

The domain rules are specified in a IF-THEN format similar to an expert system language [17]. The premise of a rule is a list of conditions that must be satisfied; the conclusion states the task that is accomplished once all of the conditions in the premise are satisfied. Each condition is a boolean expression comprised of either a single predicate, assumed to return a boolean value, or a pair of predicates and/or constants joined by a boolean operator. Variables can be used as parameters to a predicate. The following domain rule encodes the policy that if a purchase of less than $1,500 is made for travel, then the purchase can be approved.

purchaseApproval:
    IF PurchaseAmount(Purchase) $\leq$ 1500 AND
        PurchaseCategory(Purchase) = travel AND
        Boss(Person Boss) AND
        SendCopy(Person Purchase Boss)
    THEN approvePurchase(Person Purchase)

Simple facts can also be encoded by assertions of the form *factName*: predicate(parameters). Thus the assertion *student1*: student(kelly) could be used to represent the fact that Kelly is a student.

There will be some predicates that do not require the use of other rules or facts in order to be satisfied. These are comprised of two types: actions and attributes. Actions are declared using

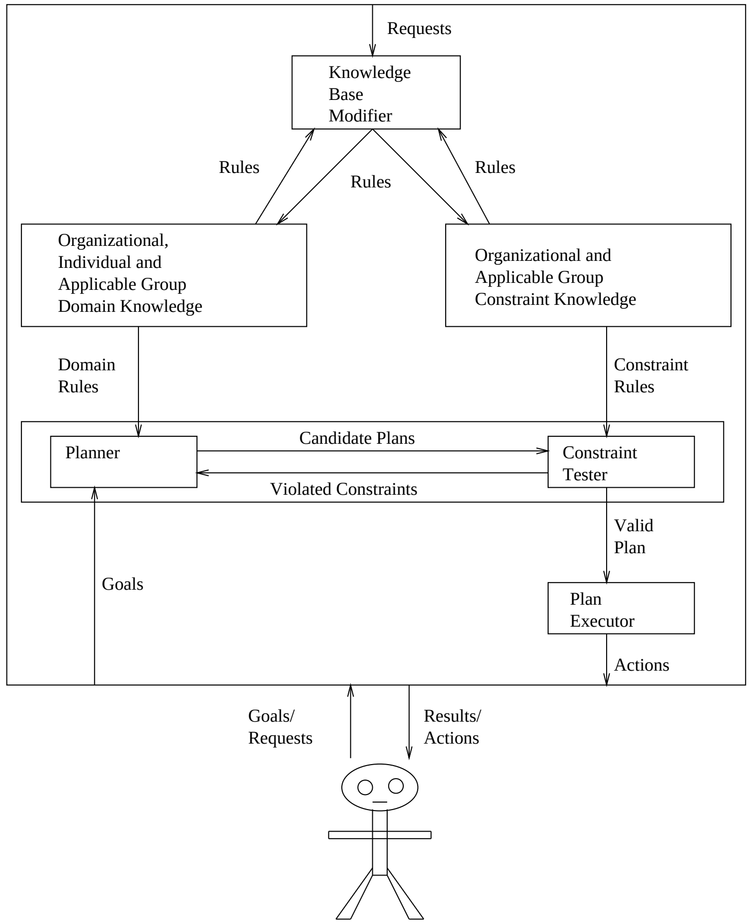

Figure 1: System architecture.

assertions of the form *Action*(PredicateName). A simple example of an action would be the sending of an electronic copy of a document to another office worker. This class of predicate could also be used in situations where the intervention of the office worker is required, such as the filling in of a form, in order to determine the values of some variables. Actions are also used to encode structured office tasks [20] and thus the rules provide a mechanism for combining structured and unstructured office work. Attributes are intended to be used to determine characteristics of the entities used in a rule. The first parameter of an attribute predicate is the entity that possesses this attribute. The remainder of the parameters are used to determine the value of the attribute. We assume that actions can have side-effects, such as the mailing of a copy of a document, while attributes cannot. This assumption ensures that an attribute can be evaluated at any time without causing interactions with office workers.

One distinction that is made between facts/rules and attributes/actions is the number of different results from successive executions of the predicate, where results is interpreted as the success/failure of the predicate, its return value, and any bindings of unbound parameters. The term *single-valued* is used to denote a predicate that, given the same parameter bindings, returns the same result every time. *Multi-valued* predicates return successive members of a (possibly infinite) set of results on successive invocations. By assumption, predicates defined by attributes and actions are single-valued, while predicates defined by facts and rules are multi-valued. For example, if we have an action that displays a partially completed form on a office worker's screen and requires the office worker to complete it, it is logical to assume that, given the exact same situation, the office worker will always enter the exact same values. Conversely, there can be several facts/rules that conclude the same predicate, with possibly different results arising from using different facts/rules.

The constraint rules are meta-constraints or meta-level rules [3] that make demands on the actions that are to be performed in a particular situation. These rules are also expressed in an IF-THEN format very similar to the domain rules. A builtin predicate, *proving*, is used to test whether the parameter to this predicate, which is itself a predicate, was proven during the planning. The conclusion specifies actions that must be present in the plan. These predicates are joined by conjunctions and negation is allowed. The following constraint rule defines the policy that when a D level manager is approving a purchase order, a copy must be sent to the manager of this manager and a copy must be sent to the accounting department.

purchaseApprovalConstraint:
  IF *proving*(approvePurchase(Manager, Purchase)) AND
    ManagerLevel(Manager, d) AND
    Boss(Manager, Boss)
  THEN SendCopy(Manager, Purchase, Boss) AND
    SendCopy(Manager, Purchase, accounting).

2.2 Individual Knowledge Bases

An IKB is composed of a set of domain rules (DR) in which we encode the office worker's problem solving knowledge, meta-level knowledge that allows the office worker to modify the DR, a list of GKBs it has access to, and the necessary planning information. The IKB's domain rules have the same format as the domain rules in the OKB.

The planning process controls the formation of an IKB's method for solving a problem. During the planning process, the planner can use domain rules and facts from the particular IKB that is involved, the GKBs that it has information about, and the OKB. This allows actions that are common to a group of office workers to be encoded in just one place (a GKB or the OKB) instead of in each IKB. This also holds true for the constraint rules. Thus the IKB appears to "inherit" both the domain rules and the constraints of the applicable GKBs and the OKB. Given the dynamically changing contents of the rule bases and the changing structure of the organization, simple static inheritance by the IKBs is not possible. Thus, the GKBs and the OKB must accept requests from the IKBs about the current contents of its DR and CR.

As noted in [2], the plans generated may be incomplete due to their dependence on information supplied by the office worker, such as values filled into a form. This can be detected in the planning process as an action predicate that has some unbound parameters. This action will provide information that is needed to continue the planning, and thus the action must be executed during the planning procedure. If this action succeeds, the planner continues attempting to create the rest of the plan.

The resulting plan contains a list of actions that must be executed, along with a list of predicates that were proved. This plan is then compared with the applicable constraints to determine if it is valid. For each constraint rule, the conditions are evaluated in much the same way as in the planner. The major difference is the *proving* predicates, which are tested by looking up the specified predicate in the list of proven predicates. If the premise does not succeed then the constraint is not applicable to the current plan. Otherwise, the conclusion of the rule is tested. For each predicate in the conclusion, if the predicate is not present in the list of actions, then this constraint is violated. If all the predicates are present in the list then the constraint is satisfied.

As in the planning, the constraint rule bases of the GKBs are scanned first, followed by the OKB. The GKBs are scanned in the order in which they appear in the IKB's list of groups. This allows the (possibly) more stringent group level policies to be tested first.

When the conclusion of a constraint rule is tested, the complete list of actions in the plan is scanned. Thus a constraint that applies to a set of predicates can be satisfied by actions that were not part of the plan for any predicates in the set. This was deliberately chosen since it seems to correspond closely to the manner in which people satisfy constraints.

If a constraint is violated, the planning procedure will be re-invoked in order to begin backtracking. The replanning will start at the last predicate in the proven list, whether or not it was referenced by the constraint. This is required in order to remain consistent with the above constraint satisfaction approach.

2.3 Knowledge Base Modification

Since office workers may want to alter their problem solving methods, there must be facilities that allow the addition, deletion, and modification of the domain rules in the OKB, GKBs and IKBs. Similarly the policies of the organization or its groups will evolve, requiring the modification of the constraint rules in the OKB and the GKBs. Thus, each of the OKB, GKBs and IKBs has meta-level knowledge describing how to modify its DR.

When modifications are made to a knowledge base, there is the possibility that it has become inconsistent. Given the undecidability of full consistency checking [14], only pairs of rules are tested for inconsistencies. Once a possible inconsistency has been detected, the inconsistent rules are reported to the office worker, who can then make necessary modifications. If the office worker does not agree that there is an inconsistency, then this will be noted to prevent detection of these rules as inconsistent in any subsequent consistency checking.

2.4 Migrating Knowledge

We will describe the knowledge migration mechanism by outlining how a domain rule is migrated from an IKB to the OKB. First, a request is given to the IKB to remove the rule. (This avoids the redundancy caused by having multiple copies of the rule.) Then, a request is given to the OKB instance to add the rule. Finally, a request is given to the OKB instance to add the appropriate constraint(s) specifying that, in the situation where the migrated rule applies, the actions taken by the office worker must include those specified in the rule. This last request is required in order to enforce the new, organizational problem-solving method on the office workers. If an office worker has different methods for solving this problem then the constraints will ensure that these methods will be used only if they are a superset of the actions defined by the organization's rules.

It is also possible that in some cases, rules may be migrated in order to become "default" rules for situations where an office worker does not have knowledge of how to solve a certain problem. This is achieved by the same procedure as above except that the corresponding constraint rule is not created.

In addition to migrating a domain rule from an IKB to the OKB, the same procedure can be used to migrate either a domain rule from an IKB to a GKB, a domain rule from a GKB to the OKB, or, with the obvious modifications, a constraint rule from a GKB to the OKB.

In order to assist the office worker in the task of determining which rules should be migrated from the office worker to the organization, some of this decision making process could be encoded using criteria such as the success ratio of a rule or the similarity of rules in many of the IKBs, but any decisions should be confirmed by an external agent. To aid in this process, a query facility could be provided that would allow authorized office workers to retrieve rules and/or complete plans along with some indication of the success of the rules. It is also conceivable that the plans which are used in certain situations could be recorded. This could then be used to suggest rule re-orderings that would improve the efficiency of the planning.

3 IMPLEMENTATION

A prototype system that supports knowledge migration has been implemented in Smalltalk [6] under version 4.2BSD of the UNIX operating system [15], on Sun workstations [13]. We chose Smalltalk as our implementation environment, rather than an existing expert system shell, because we wanted to evaluate the effectiveness of object-oriented systems in supporting knowledge-intensive applications. In addition, the object-oriented approach appears to naturally model the office domain and provides excellent encapsulation facilities that separates the implementation of an object from its functionality. This feature facilitates the building of extendible systems.

In order to implement the required facilities, we defined object classes that correspond to the organization (OKB class), group (GKB class) and individual (IKB class). As stated in the previous section, an OKB is simply a distinguished GKB to which all the individuals must belong. In order to distinguish the OKB from the GKBs, there is a separate OKB class that is defined as a subclass of the GKB class, but the OKB class does not add to the behaviour defined in the GKB class.

The GKB class defines instance variables that represent the domain rules and constraint rules, along with methods that contain some of the meta-knowledge that is necessary to modify these rule bases. This knowledge describes the mechanisms that initiate the addition or deletion of rules, but it does not make any assumptions about the actual mechanisms used to implement the rule bases. The remaining meta-knowledge is encoded in the behaviour of the object classes that comprise the rule bases, rules, and rule components.

A rule is translated into a group of objects that represents it. These objects are instances of the class that corresponds to the portion of the rule that the object represents (e.g., a BooleanCondition instance represents a boolean condition in the premise of a rule). Each object has methods that encode the planning, validation, and execution mechanisms that are appropriate for that object. To illustrate how rule bases are implemented in our system, Table 1 shows the major object classes used to encode a domain rule base.

The IKBs are represented by instances of the object class IKB. The IKB class defines instance variables for the domain rules and the list of its parents. This parents list is an ordered collection of GKB instances and an OKB instance. There are also methods defined to provide the necessary interaction with the office worker and to control the planning and execution procedures.

The planning algorithms correspond to a simple backward chaining, depth-first search similar to that used in Prolog [1]. Generally, each component of a rule has methods that encode the mechanisms by which the component is satisfied during planning. During planning, each constraint rule is tested against the current plan in order to determine whether the plan violates that constraint. The implementation of the constraint testing procedures is quite straightforward, given the planning procedures and the data structures used. After the plan is formed and validated it must be executed. This is a simple task since we make the assumption that the actions in a plan will always succeed. Given this assumption, it is possible for the planning routines to return the plan as their result, allowing a separate routine to invoke execution of the plan. If this assumption had not been made, then the notions of execution and planning become intertwinned since the failure of an action would lead to replanning [2, 19].

Given the simple structure of the rule bases, the algorithms to add, delete, modify, and migrate a rule are quite simple and will not be discussed here. In the process of adding or modifying a rule, a consistency error may be detected. When this happens, the office worker is presented with an

Class Name	Description
DRkb	Represents a domain rule base
DRentry	Parent class of all entries in a domain rule base
DRrule	Subclass of DRentry that describes domain rules
DRfact	Subclass of DRentry that describes domain facts
Condition	Parent class of the various forms of a domain rule condition
BooleanCondition	Subclass of Condition that represents a boolean condition
Action	Subclass of Condition that represents an action predicate
SpecialAction	Subclass of Action that represents an action executed during planning
NormalAction	Subclass of action that represents an action not executed until a complete plan is formed
Attribute	Subclass of Condition that represents an attribute predicate
NegatedCondition	Subclass of Condition that represents a negated condition
Predicate	Contains a predicate name and parameter list
ParameterList	An OrderedCollection of ParameterPairs
ParameterPair	Contains a parameter name and value

Table 1: Domain rule base object classes.

indication of the rules involved. If the office worker notices that there is a problem with the two rules, then a rule base browser is automatically invoked and the offending rules can be modified.

The user interface that has been implemented is basically an "expert's" interface since it is modeled after the standard Smalltalk programming interface. Various browser tools are provided in order to allow the browsing and modification of the rule bases.

4 CONCLUSIONS

In this paper, we outlined an approach for supporting knowledge migration. This approach structures individual, group, and organizational knowledge bases so that they can be modified dynamically and allows the planning process to use multiple bodies of knowledge, along with the group and organizational constraints, during the problem-solving process. Our approach provides a method by which these bodies of knowledge and constraints may be explicitly separated and yet combined in the desired ways. This structure also allows different individuals to define their own methods for solving a problem, independent of one another and allows this individual knowledge to be migrated to become either the organization-wide, or group-wide, standard. The group and organization knowledge bases provide shared access to common knowledge, and thus can eliminate, or at least reduce, rule redundancy. Also, bodies of specialized knowledge can be easily added and made available to a selected subset of the individuals in the organization.

References

[1] Clocksin, W.F., and Mellish, C.S., *Programming in Prolog, 2nd ed.* Springer-Verlag, Berlin, 1984.

[2] Croft, W.B., "Representing office work with goals and constraints," *Proc. IFIP WG8.4 Workshop on Office Knowledge: Representation, Management and Utilization*, 1987, pp. 13-18.

[3] Davis, R. and Buchanan, B.G., "Meta-level knowledge," Chapter 28, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Shortliffe, E.H. and Buchanan, B.G., eds., Addison-Wesley, Reading, MA, 1984.

[4] Fikes, R.E., and Henderson, D.A., Jr., "On supporting the use of procedures in office work," *Proc. 1st Annual Natl. Conf. on Artificial Intelligence*, 1980, pp. 202-207.

[5] Fikes, R.E., and Nilsson, N.J., "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, **1**(1), 1971, pp. 189-208.

[6] Goldberg, A., and Robson, D., *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, Reading, MA, 1983.

[7] Kornfeld, W.A., and Hewitt, C.E., "The scientific community metaphor," *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-11**(1), 1981, pp. 24-33.

[8] Lochovsky, F.H., "Managing Office Tasks," *Proc. IEEE Office Automation Symp.*, 1987, pp. 247-249.

[9] Raatz, S., and Drastal, G., "Rule interaction in expert system knowledge bases," *Proc. 6th Canadian Conf. on Artificial Intelligence*, 1986, pp. 188-193.

[10] Shortliffe, E.H., and Buchanan, B.G., eds. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, MA, 1984.

[11] Shortliffe, E.H., and Buchanan, B.G., "Knowledge engineering," Chapter 7, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Shortliffe, E.H. and Buchanan, B.G., eds., Addison-Wesley, Reading, MA, 1984.

[12] Stefik, M., "Planning with constraints (MOLGEN: Part 1)," *Artificial Intelligence* **16**(2), 1981, pp. 111-139.

[13] *The Sun Workstation Architecture.* Sun Microsystems Inc., Mountain View, CA, 1986.

[14] Suwa, M., Scott, A.C., and Shortliffe, E.H., "Completeness and consistency in a rule-based system," Chapter 8, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Shortliffe, E.H. and Buchanan, B.G., eds., Addison-Wesley, Reading, MA, 1984.

[15] Thompson, K., and Ritchie, D., "The UNIX time-sharing system," *Bell Technical J.* **57**(6), 1978, pp. 1905-1929.

[16] Tsichritzis, D., Fiume, E., Gibbs, S., and Nierstrasz, O., "KNOs: KNowledge acquisition, dissemination, and manipulation Objects," *ACM Trans. on Office Inf. Sys.* **5**(1), 1987, pp. 96-112.

[17] van Melle, W., Shortliffe, E.H., and Buchanan, B.G. "EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems," Chapter 15, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Shortliffe, E.H. and Buchanan, B.G., eds., Addison-Wesley, Reading, MA, 1984.

[18] Wilkins, D.E., "Domain independent planning: Representation and plan generation," *Artificial Intelligence* **22**(3), 1984, pp. 269-301.

[19] Wilkins, D.E., "Recovery from execution errors in SIPE," *Computational Intelligence* **1**(1), 1985, pp. 33-45.

[20] Woo, C.C., and Lochovsky, F.H., "Supporting distributed problem solving in organizations," *ACM Trans. on Office Inf. Sys.* **4**(3), 1986, pp. 185-204.