POISE: AN INTELLIGENT INTERFACE FOR PROFESSION-BASED SYSTEMS

W. Bruce Croft
Lawrence S. Lefkowitz
Victor R. Lesser
Karen E. Huff

Computer and Information Science Department
University of Massachusetts, Amherst, MA 01003

TR 82-19

This research was supported by a Digital Equipment Corporation External Research Grant.

ABSTRACT

POISE is an interface designed for systems which provide a set of tools to aid professionals in the execution of their tasks. In particular, POISE provides a means of defining the goals of typical tasks (or procedures) and how tasks can be carried out using the tools. The procedure descriptions are used both to interpret user actions and to execute tasks. The combination of interpretation and goal-based planning gives POISE the following capabilities: maintaining agendas of activities, proposing alternative courses of action, automating tasks, correcting local and global errors, abstracting user actions and invoking high-level tasks. The current implementation of POISE is described using an example from an office environment.

INTRODUCTION

Many systems currently available or being developed provide their users with a set of tools designed to help them accomplish tasks in their particular environment. The tools provided and the type of users involved will depend heavily on the particular profession supported by the system. Some examples of these profession-based systems are office information systems, software development environments and medical information systems. The tools differ widely in their sophistication, from simple electronic mail facilities to expert systems which can provide information on very specialized topics (for example, a medical diagnosis tool). A common feature of current systems is that tools only deal directly with very primitive tasks. Higher level tasks that may be more relevant to the professional's functions and which may require a number of tools cannot be supported by the system. For example, office systems provide tools such as editors and mail systems whereas the office worker is more concerned with tasks such as decision making or processing orders in a specific way.

We propose that an interface for a profession-based system should include a model of how the tasks in a particular environment can be carried out using the tools. This model would include standard ways of accomplishing various goals as well as methods used by individual users. This paper describes the design and current implementation of a profession-based system interface called POISE (Procedure Oriented Interface for Supportive Environments) that includes task models. This approach to the design of an interface has the additional advantage of providing a common framework within which the user can be introduced to a large variety of tools.

POISE interprets a user's actions in the context of a model of possible actions represented as a hierarchy of procedure descriptions (or plans). The procedure descriptions specify typical combinations of tool instantiations and the goals of the actions carried out by the tools. The ability to combine interpretation (or recognition) of actions using procedure descriptions, and planning using the descriptions and goals gives POISE great flexibility.

The POISE approach can be used in any environment that includes standard or typical procedures for carrying out tasks. A good example of this type of environment is the office where much work has been done on the analysis of standard procedures and on systems designed to support these procedures [ELLI80, ZISM77]. POISE is currently being used in an office information system and in the less structured environment of software development [HUFF82]. A system based on a similar approach was used as in interface to an operating system [SHRA82].

The approach to interface design used in this project is related to the Consul system [MARK81] and the Programmers Apprentice [WATE82]. However, in contrast to these systems, POISE contains descriptions of user tasks that involve complex sequences of actions that may require a number of the available tools, and it attempts to recognize and act on partial instantiations of the appropriate plans. A central theme of the POISE approach is to combine the efficiency of a procedure-based representation with the flexibility of a goal-based representation.

The basic POISE architecture is shown in Figure 1. The semantic database contains descriptions of the tools and the objects referred to by the procedures. The model of a user's state includes partial instantiations of procedure descriptions with parameters derived from specific user actions and instantiations of the database objects.

Being able to do interpretation and planning in the context of the semantic database and the partially instantiated set of hierarchically related procedure descriptions gives POISE the following capabilities:

a. *Planning used to propose actions.* By using the goals and sequences of actions specified in the procedure library, POISE can describe alternative courses of action to a user who is uncertain of how to carry out a task. It can also provide default values for incompletely specified actions.

b. *Planning used for task automation.* Whereas some procedures require human interaction, other tool instantiations in procedures can be invoked by the system.

c. *Propagating constraints to correct local and global errors.* Specific user actions apply constraints to the general procedure descriptions. By following the implications of a user's actions through a procedure, the system can recognize actions that, though syntactically correct, appear inappropriate in the context of what the user is trying to do. A user may, for example, destroy some entity that will be needed to complete the task. POISE could warn the user of such potential problems before an actual error occurs.

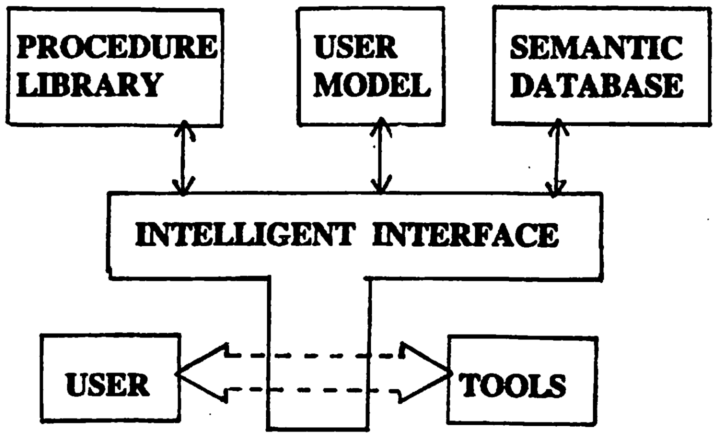

Figure 1: The POISE system.

d. *Abstracting user actions*. Since the procedures represented in POISE are specified hierarchically (i.e. procedures located further up in a hierarchy represent more abstract tasks), the system is not only able to recognize a user's action, but is also able to interpret it as being a part of some higher level procedure and thus understand the action at a more abstract level. This capability is used in summarizing and predicting activities.

e. *Agenda maintenance*. The system can keep track, over a number of terminal sessions, of a user's activities. At any time, the users can ask for agendas of their activities which the system will present as partially completed procedures.

f. *Providing a higher-level interface*. Because POISE contains a hierarchy of procedures at different levels of abstraction, the user can interact with the system at various levels. For example, the user can invoke abstract tasks which do not correspond to actual tools.

The features mentioned above will use a natural language interface that is currently under development. This interface will be for user requests to POISE and for generating natural language descriptions of the current state [MCDO81].

PROCEDURE DEFINITION

In order to represent the possible sequences of concurrent actions in a procedure, we are using a modified version of an Event Description Language [BATE82]. Figure 2 presents an example of a procedure description. The algorithmic syntax of the procedure is specified by the IS clause, refined by the COND clause and has its parameters defined by the WITH clause. The conditions required for a procedure to begin are specified by the PRECONDITION clause while the goals satisfied by a procedure are contained in the SATISFACTION clause.

The IS clause of the procedure definition provides a very precise way of describing the standard algorithm for accomplishing a task in terms of other procedures and primitive operations (tool invocations). The sequence of constituent procedures is specified using the operators Catenation ('), Alternation (|), Shuffle (#), Optional ({}), Plus (+) and Star (*). The Catenation operators specify the exact temporal ordering of two procedures. If only one of two procedures is to occur, the Alternation operator is used. Shuffle permits the interleaving of the components of two procedures in any order. The Optional operator is used to specify that a procedure may or may not occur. Plus operators allow procedures to occur one or more times, while Star, the closure of Plus, indicates zero or more occurrences.

Constraints may be placed upon the values and relationships of attributes of procedures. These constraints are specified by conditions that must be met in order to have a valid instantiation of the procedure. The COND clause is used to

```
PROC     Purchase_items

DESC     (Procedure for purchasing items with non-state funds.)

IS       (Receive_purchase_request
          ' (Process_purchase_order | Process_purchase_requisition)
          ' Complete_purchase)

COND     (and  (or  (eq  Process_purchase_order.Amount   Receive_purchase_request.Amount)
                    (eq  Process_purchase_requisition.Amount   Receive_purchase_request.Amount))
               (or  (eq  Process_purchase_order.Items   Receive_purchase_request.Items)
                    (eq  Process_purchase_requisition.Items   Receive_purchase_request.Items))
               (or  (eq  Process_purchase_order.Vendor   Receive_purchase_request.Vendor)
                    (eq  Process_purchase_requisition.Vendor   Receive_purchase_request.Vendor))
               (or  (eq  Process_purchase_order.Amount   Complete_purchase.Amount)
                    (eq  Process_purchase_requisition.Amount   Complete_purchase.Amount))
               (or  (eq  Process_purchase_order.Items   Complete_purchase.Items)
                    (eq  Process_purchase_requisition.Items   Complete_purchase.Items))
               (or  (eq  Process_purchase_order.Vendor   Complete_purchase.Vendor)
                    (eq  Process_purchase_requisition.Vendor   Complete_purchase.Vendor)))

WITH     ((Purchaser   Receive_purchase_request.Purchaser)
          (Amount      Receive_purchase_request.Amount)
          (Items       Receive_purchase_request.Items)
          (Vendor      Receive_purchase_request.Vendor))
```

Figure 2: An external procedure specification.

describe these constraints.

Attributes of a procedure are defined in terms of attributes of its constituent procedures. This information, in turn, may be used by (or provided by) higher level procedures. These attributes of a procedure are defined by the WITH clause.

The POISE formalism also contains a description of the state of the environment that must exist in order for the procedure to begin. The PRECONDITION clause specifies the set of conditions that must be true to start a procedure.

Upon completion of a procedure, certain conditions must be satisfied. This information serves both as an aid to the planner and as an alternate means of recognizing the completion of a procedure. The SATISFACTION clause specifies these conditions.

AN EXAMPLE

To demonstrate the functionality of the current version of POISE, this section describes the system being used for processing a purchase order. Figure 3 presents a portion of the procedural hierarchy for this task. Figure 4 shows part of POISE's interpretation of a set of user actions in this domain.

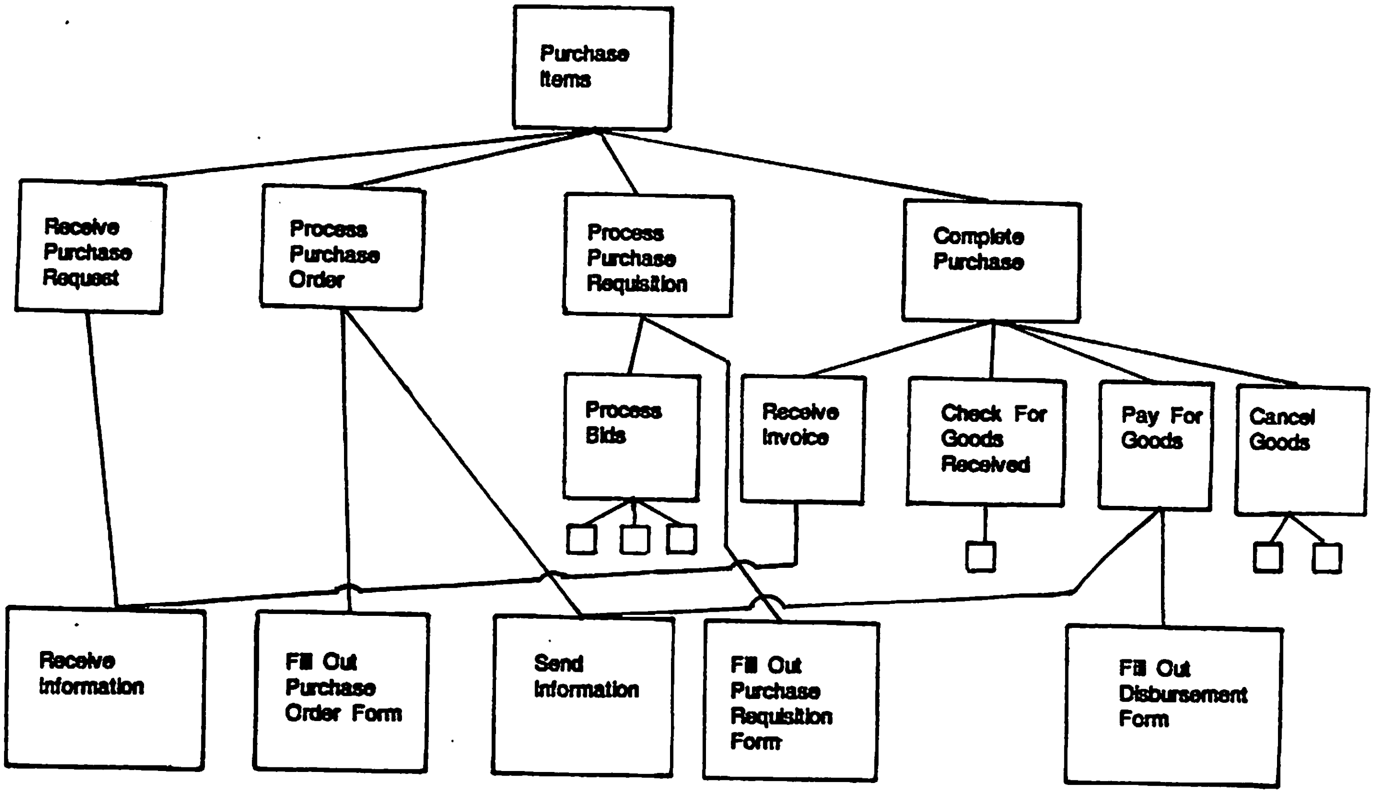

Figure 3: Procedural hierarchy.

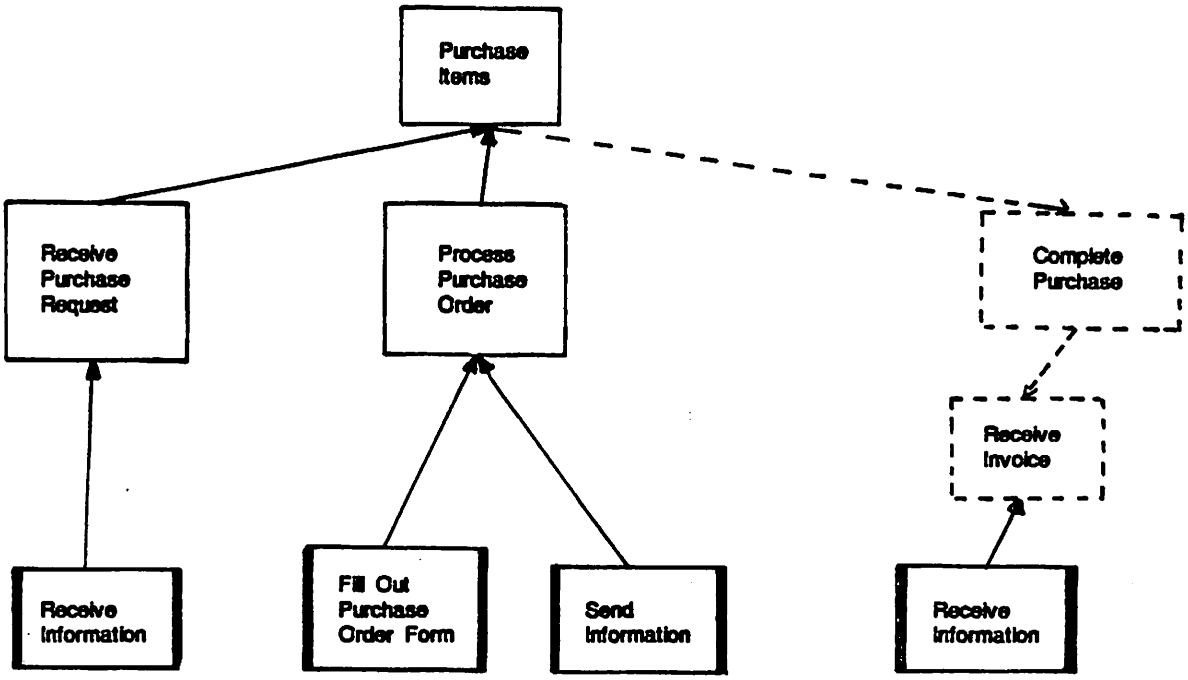

Figure 4: Interpretation of user actions.

First, the external representation of the procedure descriptions is presented to POISE. (See Figure 2 for a description of the "Purchase_Items" procedure.) POISE generates an internal form of the descriptions, verifying the existence of constituent procedure descriptions and generating rules to enable the propagation of procedure attributes. (Figure 5 shows part of the rule set for "Purchase_Items".)

When a user action occurs, POISE creates an instantiation of the procedure type(s) that correspond to that action. If the user receives information (e.g. via electronic mail), POISE creates a Receive_Information instantiation and fills in any available attributes of the procedure (e.g. source of the information, format, description, etc.). This monitoring facility is being developed to interact with an existing set of office tools. Currently, it is simulated by entering the user actions interactively from a file.

Upon creation of an instantiation, POISE attempts to integrate it into its existing model of the user's actions and goals. The instantiated procedure may be a part of a higher level task. If the procedure can begin a task (i.e. is the first step in some other procedure), the Abstractor wil create an instantiation of this higher level procedure. For instance, if the user receives information of the type "message" with a description "purchase request" (and POISE creates a corresponding instantiation of Receive_Information), the Abstractor recognizes that this is a legitimate first step in the Receive_Purchase_Request procedure. Thus, an instantiation of Receive_Purchase_Request may be created and any available attributes will be propagated up to it using the rules generated earlier.

Some of the constituent procedures of an instantiated procedure may have completed while other constituents are not yet finished. Processing a purchase order, for example, consists of filling out a purchase order form and sending information (the form) to the purchasing department. If the form has been filled out but not yet sent, POISE will contain a pending instantiation of the Process_Purchase_Order

```
(if    (selector Process_purchase_order Amount)
         (set Receive_purchase_request Amount
                 (selector Process_purchase_order Amount)))
(if    (selector Receive_purchase_request Amount)
         (set Process_purchase_order Amount
                 (selector Receive_purchase_request Amount)))
                          .

                          .

                          .
(if    (selector Receive_purchase_request Purchaser)
         (set self Purchaser
                 (selector Receive_purchase_request Purchaser)))
(if    (selector Receive_purchase_request Amount)
         (set self Amount
                 (selector Receive_purchase_request Amount)))
(if    (selector Receive_purchase_request Items)
         (set self Items
                 (selector Receive_purchase_request Items)))
(if    (selector Receive_purchase_request Vendor)
         (set self Vendor
                 (selector Receive_purchase_request Vendor)))
```

Figure 5: Constraint propagation rules.

procedure. If the user sends information (with the appropriate attribute values), the Connector portion of the system will attach the resultant Send_Information instantiation to the Process_Purchase_Order instantiation that is expecting it.

Alternatively, POISE may make predictions from pending procedure instantiations. The expectations of pending instantiations (here, the Process_Purchase_Order expects a Send_Information) may be used to create predicted instantiations which may, in turn, be used to focus the system's processing.

There are errors that POISE can detect (and sometimes correct) based on its model of the user's state. If, for instance, the user receives a purchase request and proceeds to fill out a purchase order form corresponding to the request but fills in an incorrect value on the form (such as a different amount for the purchase than was stated in the request), POISE can warn the user of this discrepency.

Occasionally, the user will carry out a task in a way that POISE does not expect. For example, if the purchase request was received in person rather than by electronic mail, the system would not be able to monitor this action. However, by utilizing the information provided bu the Satisfaction clause of the procedure description to check if the objectives of the apparently missing action have been satisfied, POISE may infer that the action has occurred.

CURRENT IMPLEMENTATION STATUS

An initial version of the POISE system has been implemented and is running in CLISP under VAX/VMS. The current system is capable of interpreting a user's actions in the context of a collection of procedure descriptions. The procedure descriptions are presented to the POISE system in an augmented Event Description Language format. The procedure reader translates these into an internal representation and generates constraint propagation and checking rules from the information provided by the WITH and COND clauses of the procedure descriptions. These rules are used by the constraint propagation mechanism to pass information around the procedure instantiation hierarchy and by the abstraction and prediction modules to assist in the interpretation process.

Upon recognizing a user's action, POISE creates an instantiation of the procedure template corresponding to it and fills in any information provided by the action. System actions based on this instantiation, such as connecting it to a more abstract procedure instantiation or making a prediction from it, are enqueued.

The scheduler module implements a focusing scheme. It is used to control the selection of system activities from the agenda of pending actions. The scheduler may be tuned to emphasize certain types of actions or to concentrate on particular areas of the procedure hierarchy.

The interpretation mechanism abstracts upwards in the procedure hierarchy from an instantiation as well as predicting the next (lower level) steps in an instantiation. The abstraction process consists of determining what higher level procedures an instantiation can be part of and connecting the instantiation to them. The constraints are checked (using the rules generated by the procedure reader) to verify that the interpretation is semantically valid. If the instantiation is the first step in the higher level procedure, the system will first create an new higher level instantiation and then connect up to that. For prediction, POISE determines the set of possible next constituent procedures for an instantiation and generates the necessary predictions. Using constraint propagation, attribute values are filled in on the predicted instantiations.

The objects manipulated by the procedures (as well as those objects that do the manipulating) are described in a KLONE-like semantic database. The development of this database and its integration into the POISE system is currently underway.

A simple debugging facility exists and is used to examine the state of the interpretation. It provides a view on particular instantiations, their connections to other instantiations, and the scheduling agenda.

CONCLUSION AND FUTURE RESEARCH ISSUES

The current version of POISE demonstrates that in an environment such as the office where many of the actions in procedures can be readily identified, the user's goals can be quickly inferred from a partial sequence of actions. This is accomplished by focusing on likely interpretations using domain-independent heuristics. By propagating constraints derived from the user's actions, the system can then provide significant assistance in the execution of tasks and the correction of errors. It remains to be seen whether more domain-specific knowledge will be required in other environments. For example, in the software development environment, the actions tend to be more ambiguous and the number of potential interpretations can grow very rapidly.

Another major issue for a future version of POISE is to provide an interface for user creation and modification of procedures. This interface will be a crucial part of dealing with the changing environments typical of profession-based systems. Two possible approaches to this problem are constrained example generation [RISS80] and planning. Users may be able to modify procedures by modifying examples of procedures. A planning mechanism which is independent of procedure hierarchy may be able to construct new procedures using the preconditions and goals of basic procedures and by interacting with the user to fill in gaps.

Acknowledgement

The authors wish to acknowledge the work of Norman Carver, Dan McCue and Susan Lander on the implementation of the POISE system.

References

BATE82 Bates, P.C.; Wileden, J.C. "EDL: A basis for distributed system debugging tools". *International Conference on Systems Science*; Hawaii, 1982.

ELLI80 Ellis, C.A.; Nutt, G.J. "Office information systems and computer science". *ACM Computing Surveys*, 12: 27-60; 1980.

HUFF82 Huff, K.E.; Lesser, V.R. "Knowledge-Based Command Understanding: An Example for the Software Development Environment". COINS Technical Report 82-6, University of Massachusetts; 1982.

MARK81 Mark, W. "Representation and Inference in the Consul System". *IJCAI-7*, 1981.

MCDO81 McDonald, D.D. "Natural Language Generation as a Computational Problem: an introduction". in *Computational Models of Discourse*, The MIT Press, Cambridge, Massachusetts, 1983.

RISS80 Rissland, E.L.; Soloway, E. "Overview of an Example Generation System". *Proceedings of the AAAI*, Stanford, CA; 1980.

SHRA82 Shrager, J.; Finin T. "An Expert System that Volunteers Advice". *Proceedings of the AAAI*, Pittsburgh, PA; 1982.

WATE82 Waters, Richard "The Programmers Apprentice: Knowledge Based Program Editing", *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 1, January 1982.

ZISM77 Zisman, M.D. *Representation, Specification and Automation of Office Procedures*, Ph.D. Dissertation, Wharton School, University of Pennsylvania, 1977.