TASK SUPPORT IN AN OFFICE SYSTEM

W. Bruce Croft

Lawrence S. Lefkowitz

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

ABSTRACT

A major goal of an office system is to support the tasks that are central to the office functions. Some office tasks are readily implemented with generic office tools, such as calendars, forms packages and mail. Many tasks, however, involve complex sequences of actions that do not all correspond to tool invocations and, instead, rely on the problem-solving ability of the office workers. In this paper, we describe a system (POISE) that can be used to both automate routine tasks and provide assistance in more complex situations. The type of assistance provided can range from maintaining a record of the tasks currently being executed to suggesting possible next steps and answering natural language queries about the tasks. The POISE system uses both a procedure-based and a goal-based representation of the tasks to achieve efficiency and flexibility. The mechanisms used by POISE are described with example procedures from a university office.

Categories and Subject Descriptors: H.4.1 [Information Systems Applications]: Office Automation; H.4.2 [Information Systems Applications]: Types of Systems—*decision support*; I.2.1 [Artificial Intelligence]: Applications and Expert Systems—*office automation*; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods—*representations (procedural and rule-based)*

1.0 INTRODUCTION

The tools in current office systems are designed to carry out simple tasks that are common to most offices. For example, tasks such as communication, time management and document production are supported by the electronic mail, calendar and text editor tools. It has been pointed out that this level of task support is only of limited effectiveness in addressing the problem of office productivity [12, 13]. A more effective system would support higher-level tasks that are directly related to the goals or functions of the office. This type of task often involves decision-making, complex sequences of actions, and interaction with a number of other people. The crucial feature of these tasks is that, unlike the simple tasks mentioned previously, they more closely implement a particular office's functions and must be very office-specific. It will not be possible to provide a generic set of tools to directly support all office tasks. However, it should be possible to "customize" an office system by providing a means of describing the high-level tasks and how they can be supported with the existing set of tools.

For example, the task of processing orders in a particular company could be described in terms of invocations of generic forms and mail tools, together with actions and decision points that have no corresponding generic tools. On the basis of this task description, the system would, in effect, provide a "virtual" tool that could support the order processing task. This distinction between virtual tools and tools that are actually implemented in the office system has similar advantages to database systems that distinguish between logical and physical descriptions of data. If a new generic office tool (or set of tools) were provided, much of the task descriptions need not change, but the way in which they are supported will. The part of the task descriptions that will change will be the mapping between steps in the tasks and the tools that implement them.

Descriptions of office tasks are usually called *office procedures*. Although the name implies that the activities described are very routine and structured, office procedures can also be used to describe more unstructured, management-level tasks. Office procedure formalisms have been used to define and analyze offices [7, 1], and to automate office tasks [17]. A description of a task using a procedure formalism is subject to change and represents only a standard or typical way of carrying out that task. A system that uses office procedures to support tasks must provide mechanisms to deal with these limitations of

procedures [9].

In this paper, we shall describe a system that supports high-level office tasks with generic office tools. The system incorporates an office procedure formalism that is used to describe the tasks and their implementation using the available tools. The system acts as an intelligent assistant to the office system user by providing, for example, task automation, agendas of activities and information for decision-making [4]. A natural language interface is used to access information about the status of active tasks.

In the following section, we shall outline the likely user requirements for a task support facility. These requirements will be used to evaluate the capabilities of the proposed system. Other systems designed for task support are also discussed. Sections 3 and 4 contain an overview of the task support system (called POISE) and the procedure formalism on which it is based. In sections 4 and 5, we develop examples of task description and support using the POISE system, including the use of the natural language interface. Section 6 discusses two important POISE system components currently under development. These components are the interface for user specification and modification of procedures and the planning mechanism.

2.0 USER REQUIREMENTS FOR TASK SUPPORT

The task support facility must be flexible in order to satisfy the different requirements of the users of an office system. Tasks in the office have been characterized by the amount of "structure" they contain or, conversely, by the degree of problem-solving involved in the task [3]. At one end of the scale, very structured tasks, such as producing payroll checks, can be carried out with algorithms specified in application programs. At the other end of the scale, an unstructured task, such as deciding whether to accept a merger offer from company Y, has few characteristics that can be specified algorithmically, but relies almost completely on the problem-solving abilities of the person allocated to the task. Many office tasks fall between these two extremes, having both structured and unstructured aspects. Tasks handled by clerical workers tend to be more structured than those handled by managers.

Systems designed for task support in the office have taken different viewpoints of the structure or lack of structure involved in the office tasks. Office-By-Example [18] provides a convenient interface for specifying simple algorithmic tasks such as sending out form letters when specific conditions occur in the database. Zisman's SCOOP system [17] can deal with less structured tasks but it is designed for task automation. This means that each task step has a corresponding tool invocation. For example, a step such as "Make_decision_about_X" is implemented using a "Get_response" mail message. The emphasis on automation rather than assistance restricts the flexibility of the system.

Barber's system [3] provides the office worker with a sophisticated problem solver that will handle unstructured tasks by establishing user goals and attempting to achieve those goals. The reasoning mechanisms in this system use a complex knowledge representation for describing the objects and goals in a particular domain. For example, in the domain of processing orders for furniture, the knowledge representation would include descriptions of different types of furniture, forms used for orders and the goal of successfully filling an order. The ability to approach office tasks from a goal-oriented perspective means that a task is not bound to a fixed series of actions as it would be in Office-by-Example or SCOOP. The latter systems have no mechanism for the user to tell the system that a particular task step is not appropriate for the current situation.

The POISE system, which is described in the remaining sections, is designed to support structured tasks and some types of unstructured tasks. It attempts to combine the efficiency of the procedure-oriented approach with the flexibility of the goal-oriented approach. The current version of POISE does not have a general problem-solving mechanism but, if a standard procedure is not appropriate for a given task, it can still provide assistance to the office workers who will be doing the problem-solving. As an example of this type of decision support, consider how POISE would support the fairly unstructured task of hiring new graduate students for a research project. The problem is to decide how many people can be hired and who they should be. Although it would be very difficult to write an algorithm to handle this task, most people could write a list of things that should be done. For example,

a. Check salary budget for the project.

b. Check current salary for graduate students.

c. Check space available.

d. Get a list of unfunded students and their details.

e. Advertise the positions available.

f. Hire graduate students.

These steps can be directly specified in a POISE procedure along with the goals of the individual steps and the tools that implement them. In step a, the project manager could specify the tool invocation that could retrieve the information (a database query) and how it should be displayed. Step b may involve a phone call to the department head and therefore would not have a corresponding tool invocation. POISE would, however, be able to remind the manager to get this information. Step c could specify a standard method of checking space, such as contacting the space allocation committee in the department using a specified form message, but the manager is free to use any method of carrying out this step as long as the specified goal of finding space is met. For step d, the procedure could specify to access a student database. If there is not sufficient information to automate this step, POISE could still make the database available for browsing. Some additional control complexity could be added, such as specifying that if a definite decision is made on some likely students, then mail messages advertising the position will be sent to just these students, otherwise a mail message will be broadcast to all students. This type of support is heavily dependent on a simple method of specifying and modifying procedures. We shall return to this example in section 4.

By designing POISE to support both structured and unstructured tasks, it is intended to meet the needs of a wide variety of users. The type of environment where POISE would not be useful would be one in which most tasks are carried out once and never repeated. In order to be more specific about the functionality of the task support system, a set of requirements from the user's point of view should be considered. Different users will have different requirements and a single user will also have varying requirements depending on the task being carried out. The following list of task support system features, therefore, attempts to encompass the major user requirements for an entire office.

1. *Automation of routine tasks.* Tasks or parts of tasks that are very structured should be able to be completely automated.

2. *Assistance with complex tasks.* If the user is required to perform some actions as part of a task, the system should provide assistance in the form of explanation and error detection and correction.

3. *Context switching.* The office worker will typically be handling a number of tasks concurrently. The system should facilitate switching between these tasks.

4. *Handling unusual actions.* Office workers will need to be able to perform tasks in nonstandard ways. The minimum requirement for a task support system is not to prevent these unusual actions and to allow users to directly invoke any tool. A more difficult requirement would be for the system to recognize which task is being carried out in a nonstandard way and provide assistance based on that identification.

5. *Adaptability.* Offices are dynamic and office procedures will constantly be added and modified. User-defined procedures will be an important part of the modification process. The task support system should include a mechanism for user modification and specification of procedures and a means of maintaining the consistency of related procedures.

6. *Task invocation.* Tasks should be able to be invoked by users in a similar fashion to generic tools. These tasks should be at different levels of abstraction (e.g. processing an order vs. filling out a form field).

7. *Status inquiry and explanation.* Because of the concurrent nature of office tasks, a method of describing the status of the current tasks and explaining possible next steps should be provided by the system.

8. *Handling multi-user tasks.* Many tasks are carried out through the cooperation of a number of people. The system should support and coordinate these distributed tasks by providing status communication between different nodes in a network and by incorporating a mechanism for attaching people (or roles) to parts of tasks [8].

These requirements from a basis for discussion of task support systems. The extent to which the POISE system meets these requirements is discussed in the following sections.

3.0 AN OVERVIEW OF THE POISE SYSTEM

The POISE system provides task support on the basis of hierarchies of task (or procedure) descriptions. The procedure descriptions specify the typical steps involved in the task, the tool invocations which correspond to those steps, and the goals of those steps. The ability to combine recognition of user actions and planning using the descriptions and goals gives POISE great flexibility in the type of task support it can provide.

POISE acts as an intelligent interface between the user and the tools available in an office system. A simplified diagram of the main POISE system components is shown in Figure 1. Three types of information are used by the interface. The procedure library contains the procedure descriptions. The semantic database contains descriptions of the objects used in the procedures and descriptions of the available tools. The model of a particular user's state includes partial instantiations of procedure descriptions with parameters derived from specific user actions as well as instantiations of semantic database objects.

For example, in the procedure library there could be a procedure for filling out a purchase order form. In the semantic database, there would be a description of this form, the fields in it and its relationship to other forms and fields used in the system. After a user had started to fill in a particular purchase order form, the user model would contain a partial instantiation of the "Fill_out_purchase_order_form" procedure with values derived from the actual values filled in by the user. There would also be an instantiation of the semantic database object that represents the purchase order form.

The capabilities of the POISE system can be used to address the user requirements mentioned in the previous section. The following list describes these capabilities and notes the related user requirements.

- *Planning used for task automation.* By using the goals and sequences of actions specified in the procedure descriptions, POISE can automate routine tasks (User requirement 1). It can also provide default values for incompletely specified actions.

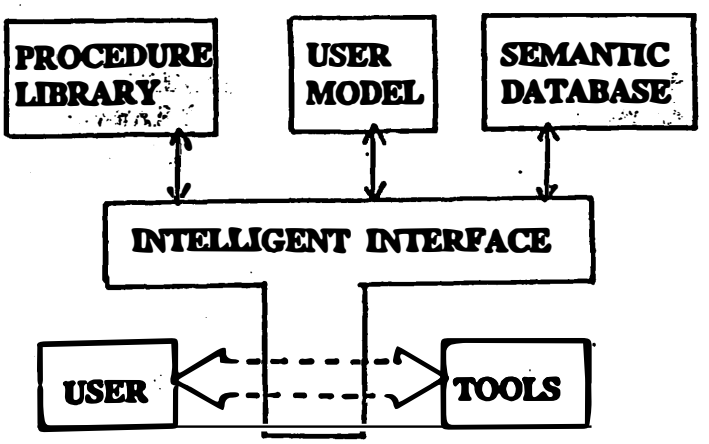

Figure 1: The POISE system.

- *Planning used to propose actions.* The same planning techniques can be used to describe alternative courses of action to a user when decision points are reached (Requirement 2).
- *Using goals to recognize actions.* By checking the goals of task steps (specified in terms of the state of the database), POISE can recognize tasks performed in nonstandard ways (Requirement 4).
- *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 (Requirement 2).
- *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 able to interpret a user's action as being 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, recognizing task invocations, and for communicating between nodes performing a distributed task (Requirements 6,7,8).
- *Interrogation and explanation using natural language.* A natural language interface is used for user requests to POISE and for generating natural language descriptions of the current state [15] (Requirement 7).

The POISE system can work in two different modes - interpretation or planning. In the interpretation mode, the user invokes tools directly and POISE attempts to recognize the user's goals in the context of the procedure library. In the planning mode, the user invokes a procedure and the system must then carry out as much of that procedure as possible. In an actual environment, POISE will make use of whichever mode is appropriate. Section 5 contains an example of POISE's operation and a description of the mechanisms that support it.

4.0 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 [2]. 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 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.

The example shown in Figure 2 is a "Purchase_items" procedure. This procedure is a typical semi-structured clerical task. The IS clause of this procedure specifies that after a purchase request has been received, either a purchase requisition or a purchase order is

```
PROC    Purchase_items

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

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

WITH    ((Purchaser      = Receive_purchase_request.Form.Purchaser)
         (Items          = Receive_purchase_request.Form.Items)
         (Vendor_name    = Receive_purchase_request.Form.Vendor_name))

COND    (for_values {Purchaser Items Vendor_name}
           (eq Receive_purchase_request.Form
               Process_purchase_order.Form
               Process_purchase_requisition.Form
               Complete_purchase.Form))

PRECONDITIONS -

SATISFACTION    (for_values (Purchaser Items Vendor)
                   (exist Complete_purchase.Form))
```

Figure 2: An example procedure specification.

processed. The task is completed by the steps involved in the Complete_purchase procedure. To get the details of the steps involved in the Complete_purchase procedure, we would have to examine the corresponding descriptions. The more detailed (or less abstract) procedures contain links to the tools available in the system. Figure 3 shows the procedure hierarchy that results from the full set of procedure descriptions for the Purchase_order example. The lowest level procedures in this hierarchy correspond (approximately) to tool invocations. The actual mapping between "primitive" procedures and tools is table-driven.

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 describe these constraints. The COND clause in the example specifies that the purchaser, vendor_name and items fields must be the same in all forms used in an instantiation of Purchase_items. The COND clause is also used to propagate attribute values during planning.

Attributes of a procedure are defined in terms of semantic database items, constant values, or 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. For example, the purchaser in the Purchase_items procedure is obtained from (or supplied to) the purchaser field of the Receive_purchase_request form.

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. The example procedure specifies that an appropriate instantiation of a Complete_purchase form must exist when the Purchase_items procedure has finished.

The PRECONDITION and SATISFACTION clauses cause demons to be attached to database objects. Upon instantiation of these objects, the demons, using parameters from the objects, find the appropriate procedure instantiations and may cause their completion.

The formalism described above is an *intermediate* language. The procedure descriptions written in this language are mapped into an internal representation by the POISE procedure reader. The language is adequate for a systems analyst but obviously unacceptable for presentation to the users of an office system. Research currently underway to develop a suitable interface is described in section 6.

The "funding of graduate students" task mentioned in the first section provides an example of a less structured task that may be specified by a managerial level person. Figures 4 and 5 show the EDL description of this task and a view of the procedural hierarchy. The IS clause for this task contains very little temporal ordering. Only obvious

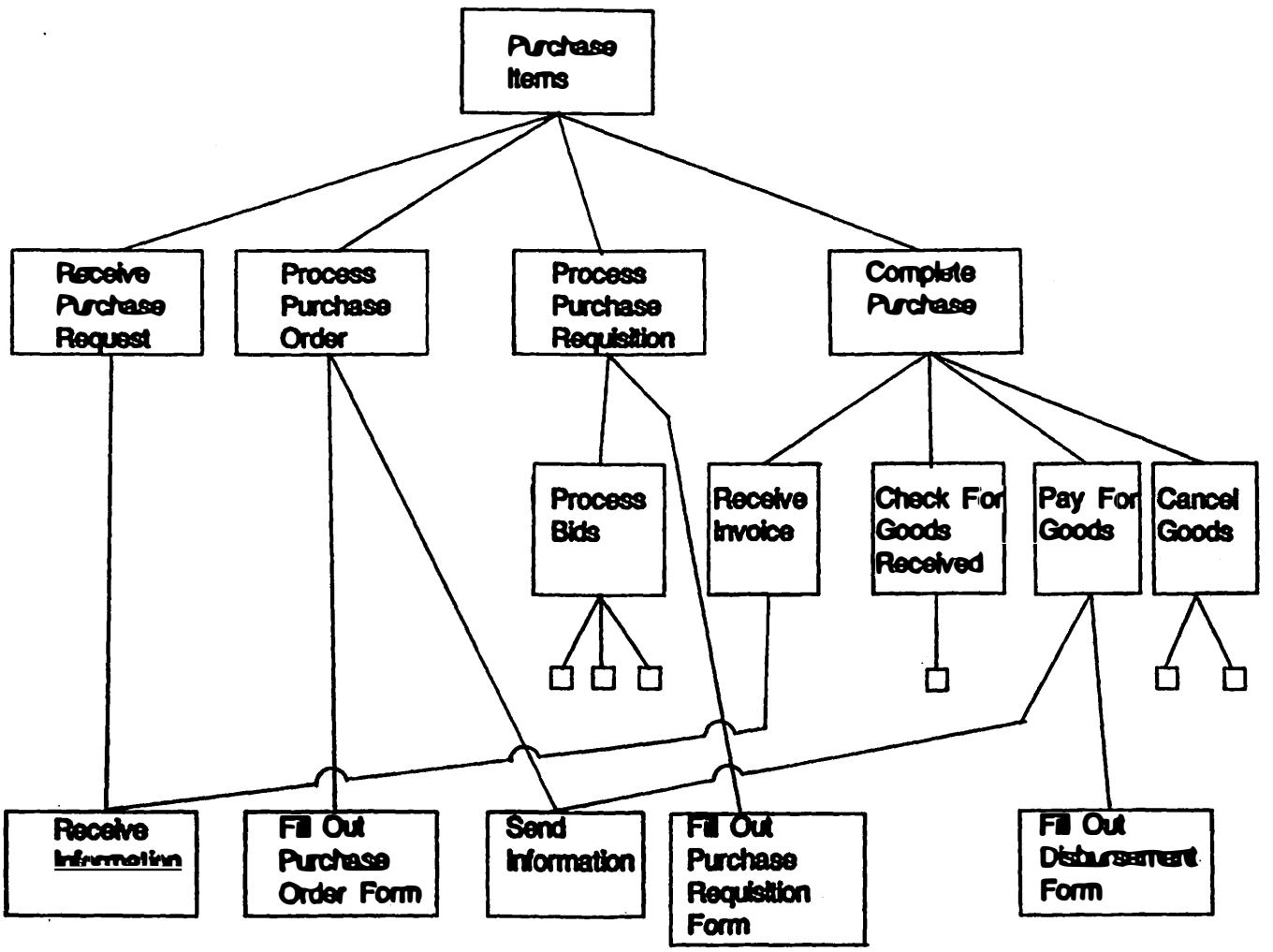

Figure 3: An example procedure hierarchy.

sequences of activities have been specified. The COND clause is described as simple rules which are equivalent to the standard COND clause shown in the last example. The rule in this example specifies that if particular students have been selected from the database, send the advertisement to them, otherwise send it to all students. The rule form of the COND clause is often more convenient from the user's point of view and provides what is in effect a simple production system for this procedure.

User-specified procedures will tend to have few levels, and Figure 5 shows that, for this example, the second and third levels of the procedure hierarchy contain the tool invocations (if the appropriate tool exists). The WITH clause simply specifies the important information for tool invocations or decision-making.

The semantic database of POISE implements a data model very similar to that described by Gibbs and Tsichritzis [10]. Part of the model for the purchasing example appears in Figure 6. The figure shows an inheritance hierarchy of objects. The data model describes office objects such as forms and mail messages as well as world objects such as

---

```
PROC   Hire_graduate_students

DESC   (Procedure for hiring graduate students for research projects.)

IS     ( (Check_salary_budget
          # Check_graduate_salary
          # Check_available_space
          # Get_graduate_students)
        (Advertise_available_positions
          # Hire_graduate_students))

WITH   (Available_students      = Get_graduate_students.Students)
       (Selected_students       = Hire_graduate_students.Students)
       (Advertisement_recipients = Advertise_available_positions.Recipients)
       (Graduate_salary         = Check_graduate_salary.Amount)
       (Budget                  = Check_salary_budget.Amount))

COND   ((if (exist Selected_students)
           then (eq Advertisement_recipients Selected_students)
           else (eq Advertisement_recipients Available_students)))
```

Figure 4: A less structured procedure.

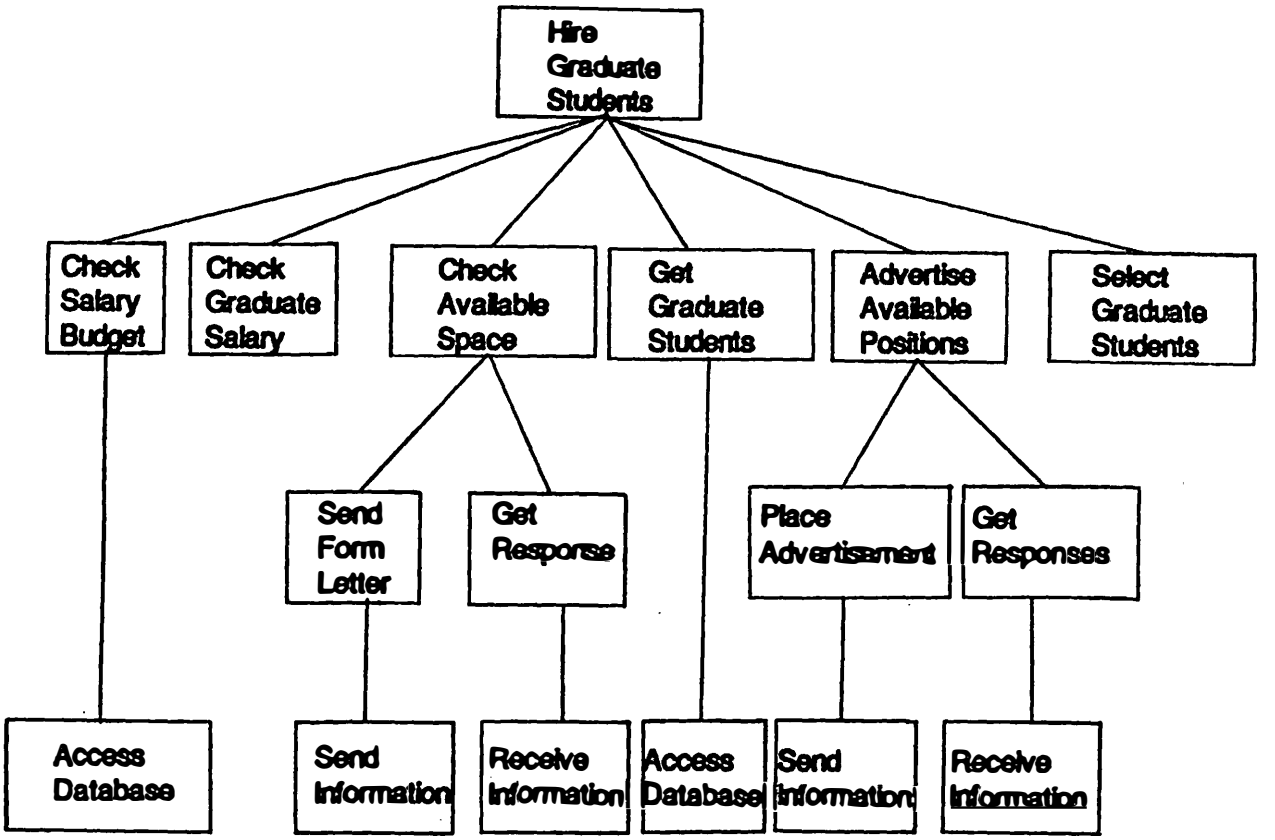

Figure 5: The procedure hierarchy for the hiring task.

vendors and items. The objects can have simple attributes such as form_number, or composite attributes such as the list of items ordered with the quantities and prices. Objects can also be attributes of other objects. The model also specifies constraints on objects and attributes. For example a constraint on purchasing forms could specify that the items named on the forms must be sold by the vendor named on the forms. The constraints make use of the knowledge specified under world objects.

The data model is implemented using a frame-based representation language (SRL)[16]. Objects map directly to frames, with attributes being described by frame slots. Composite attributes are described using other frames. For example, the items field in the purchasing forms is represented by a frame slot that contains one or more pointers to frames that contain item names, quantities and prices. The constraints are specified as procedural attachments to frame slots. The data types of attributes are either described directly using basic types (e.g. INTEGER, STRING) or indirectly by using a domain description frame. For example, the slot for the vendor name field contains a pointer to a domain description

frame. This frame describes the allowable values for this field to be the set of names in the instantiations of vendor objects. A set of standard access functions is provided that allows the POISE system or the tools (through a tool monitor) to update the database or retrieve information.

5.0 SUPPORTING AN EXAMPLE TASK

To demonstrate the functionality of the current version of POISE, this section describes POISE assisting a person who is purchasing items. The system's actions are based on the procedure descriptions and data model shown in Figures 2, 3 and 6.

At the top level of the POISE system, the user is presented with a menu of tasks (such as filling out particular forms) as well as access to the lower level tools (e.g. Mail, Forms Processing). Since POISE acts as an interface to a set of tools that were not

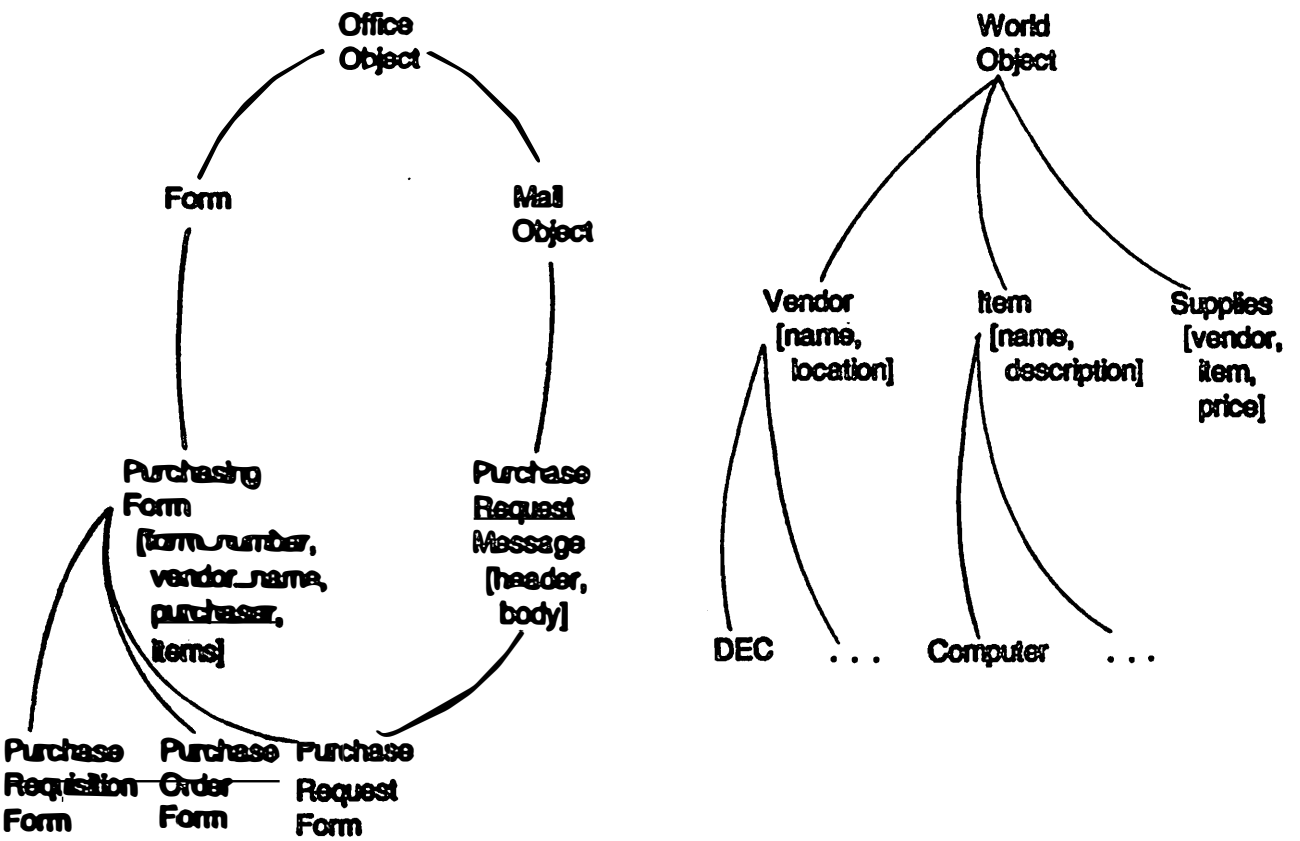

Figure 6: Simplified data model for purchasing example.

designed specifically for use with it, all access to these tools must be done through POISE menu choices. This enables the system to monitor the tool usage without requiring major alterations to existing tools. By selecting the Forms facility from a POISE menu, for example, the user can fill out various types of forms (e.g. Purchase Order, Disbursement Form). Although it appears that the user is interacting directly with the forms tool, every command and response is intercepted by POISE. If the POISE system were built as a part of an integrated office system, each system tool would be designed to provide the interface with the necessary information and functionality for monitoring and controlling tool usage.

The example task begins with a secretary receiving a request to purchase some items. POISE is then used to support the balance of the purchasing items task. The example demonstrates POISE's planning system, its error detection and correction capabilities, and its natural language help facility.

The scenario begins with the secretary reading new mail. This is done by selecting the Mail facility from the POISE menu. When a purchase request message is read, POISE can instantiate a Receive_information procedure by using tables that define the mappings between tool actions and procedures. In addition, a Purchase_request message and the corresponding Purchase_request form are instantiated in the semantic database. The fixed fields in this message reveal that it is asking the secretary to buy a desk from Steelcase for $100. POISE attaches this message to the instantiated Receive_information procedure. It then checks if this action is expected in the current context. Since there are no other activities in progress, POISE checks if this action can begin some new task. Using tables generated from the procedure hierarchy, POISE recognizes that Receive_information, with the attribute values given, can begin the procedure Receive_purchase_request, which, in turn, begins the Purchase_items task. Where multiple interpretations of an action are possible, POISE maintains all such interpretations but focuses on what is heuristically determined to be the most plausible (see section 6). If this selection later turns out to be incorrect, POISE backtracks and selects the next most likely interpretation.

Had the user's action not been a viable next step in any active procedure and not been able to begin any new tasks, POISE would have recognized this as either an error condition or as an exceptional (i.e. not specified in the procedure description) way of carrying out a task. The creation or modification of objects in the semantic database

activates demons that have been attached to these objects as a result of the conditions specified in the procedure's Satisfaction clause. This allows an alternate means of recognizing the satisfaction of the procedure by some unspecified action(s). If the step cannot be accounted for, POISE treats this action as an error and informs the secretary.

After receiving the purchase request, the secretary instructs POISE to complete the instantiated purchasing items task. From the procedure hierarchy, it recognizes the next step in the task as an instantiation of either a Process_purchase_order procedure or a Process_purchase_requisition. If the value of the purchase is less than $250, a Purchase_order form should be used. Otherwise, a Purchase_requisition form is needed. Because the value of the purchase is known from the received purchase_request, the planner is able to correctly select the Process_purchase_order procedure. This procedure, in turn, requires that a purchase_order form be created and its fields filled out. Thus, a Fill_out_purchase_order_form procedure is instantiated, and a Purchase_order form is created in the database. All the required values (e.g. purchaser, vendor_name, items) are known from the Purchase_request and can be filled in. In situations where values cannot be determined, the user is asked to supply the missing information.

POISE can then go on to invoke the Complete_purchase procedure. It has all the necessary data for checking to see if the invoice has been received and then filling out the necessary forms for paying for the received items. The procedure for checking to see if the goods have been received, however, will require that POISE ask the secretary to verify that the ordered items have actually arrived.

Rather than asking POISE to carry out a high level task (such as purchasing items), the user may wish to interact directly at the tool level. For example, upon receipt of the purchase request, the secretary may invoke the form facility and manually fill out the appropriate form. As an example of POISE's error detection and correction capability, suppose the secretary tells POISE to fill out a Purchase_requisition form rather than a Purchase_order form. POISE attempts to assimilate the secretary's action into the current state. It finds that filling out a Purchase_requisition form does not fit into the only active task (Purchase_items) nor can it begin any new tasks. However, POISE recognizes that a similar procedure, filling out a purchase order form, is expected. POISE presents this alternative action to the secretary with an explanation of the possible error in the selected

type of form. If the secretary accepts this corrected action, the forms package is used to produce a Purchase_order form.

Another example of POISE's error handling capabilities is seen when the user attempts to enter invalid values. This may occur while manually inserting data (such as filling out forms) or as a response to the planner's request for missing information. By recognizing the context in which the data fits, values that are invalid, either syntactically (e.g. alphanumeric vs. integer) or semantically (e.g. the wrong vendor name), are detected and, where possible, corrections are offered.

POISE's natural language help facility may be invoked to make inquiries about the state of active, completed or expected activities. If the secretary wished to see what activities were currently in progress, a request such as "Tell me what is currently being worked on." may be made. The natural language input module, a McDYPAR parser [6], translates this from natural language into a highly structured query requesting the current state of the system. A portion of this actual query is (info-request ... wanted: (reference ttype (pstate) tense (present))). POISE, via tables that map from entities in the parser's output to internal structures and procedure-dependent attributes, translates this into a query that seeks all currently active, plausible procedure instantiations. In this case, the only active task is Process_purchase_order which is part of Purchase_items. This response is passed onto the natural language generation facility, Mumble [15], for presentation to the secretary. Mumble has been provided with a lexicon and language structures necessary for the production of responses in the office domain. Using these, and the information provided by POISE in response to the secretary's request, Mumble formulates the output: "You are currently involved in purchasing items for Jones. You have begun to process a purchase order."

Similarly, the secretary may inquire as to the value of some entity in the current task. The question "Where are we buying the desks from?" gets parsed into a query seeking the vendor for a purchase where the item purchased was a desk. POISE maps this request into a search through those current activities that involve purchases, vendors and items, finds the one whose items are desks, and looks up the vendor. Here, the vendor is Steelcase. This is passed onto Mumble which generates "We are buying them from Steelcase."

6.0 POISE COMPONENTS UNDER DEVELOPMENT

A number of components of POISE are currently under development. Two particularly important components include the interface for procedure specification and modification, and the focusing and planning mechanisms. In this section, we will discuss these components in more detail.

6.1 Procedure specification

Three approaches to user procedure specification are being investigated. The first uses a graphical interface, the second involves modification of procedures using natural language, while the third attempts to automate the knowledge acquisition process.

The graphical interface will display procedures using pictorial conventions rather than the EDL formalism. This interface will graphically represent the IS clause rather than using the formal extended regular language syntax. In addition, a method of moving up and down the levels of the procedure hierarchy will allow the user to easily view the task at different levels of abstraction. The top level view of the procedures provided by the system is similar to a BDL diagram [11]. A procedure is shown as a nested set of boxes connected to show temporal relationships (as specified by the IS clause). Each box represents a lower level activity that can be examined in more detail by selecting it with the cursor.

The graphical interface emphasizes a "building block" approach to procedure specification with the aim of simplifying this task by providing a menu of frequently used procedures that can be used to build up more complex procedures. This should encourage top-down development of specifications and insulate the user from much of the detail of the low level tool invocations.

In the second project, the system interacts with the user to modify existing procedure descriptions. A natural language dialogue is possible because of the constrained application domain and the detailed knowledge of the procedure descriptions and possible modifications. Two important parts of this project are the development of the grammar and vocabulary needed for the task, and the generation of a model of the procedure modification dialogue. The model is used to guide the natural language parsing and generation in the user interaction. This project will eventually be combined with the graphical interface to exploit

the advantages of both media.

The final project is concerned with the acquisition of the information necessary for the specification of procedures. The goal is to automate the interview process usually required to obtain the relevant knowledge from office personnel. Actual knowledge acquisition interviews are being analyzed as a guide for the development of this system. The system is based on the assumption that the interviewer has some initial (possibly incorrect) model of the task and refines this model as the interview progresses. The interviewer may thus be considered to be iterating through a series of task models. The POISE procedure formalism is used to describe each successive model.

In order to understand the acquisition process, the differences between successive procedural models are examined to see what types of changes are being made. These changes may then be classified and the causes of each type of modification understood. This type of information will be used as the basis for the knowledge acquisition system.

An important component of the software environment for procedure specification and modification that has not yet been defined is the procedure library manager. The manager will be responsible for maintaining consistency when procedures are added or modified. This will involve checking both inter- and intra-procedure constraints. The manager will also control access to the procedures to prevent unauthorized modifications.

6.2 Focusing and Planning

POISE uses both recognition and planning modes in order to understand what task the user is attempting to accomplish and provide assistance in accomplishing it. In order to be effective, it must quickly and accurately identify the user's objectives and possible subsequent actions. A focusing mechanism that uses heuristics to narrow the range of possibilities is currently being implemented [14].

When POISE attempts to recognize partially completed procedures, it is often the case that no single interpretation of the user's actions is possible. Maintaining all possible interpretations would be expensive and could, in fact, reduce the ability of the system to provide assistance. If the set of interpretations were not narrowed down, POISE would have many possible contexts for interpreting user actions and would not be able to provide intelligent error detection/correction or planning. The focusing mechanism selects plausible

task interpretations based on heuristic information about the relative likelihood of different sequences of actions. The domain-dependent heuristics and the assumptions that can be derived from them are formalized and used in a truth-maintenance system [5]. This approach makes it possible to reason about the assumptions behind the current state of the interpretation and why they were made. When new information is acquired which invalidates the current interpretation, the system can use this reasoning ability in an intelligent backtracking scheme.

In the current POISE system, a simple planner enables tasks to be automatically carried out or completed by the system. A more sophisticated planner is under development that will enable the user to specify desired goals in addition to specifying tasks. These goals can be stated in terms of the state of objects contained in the semantic database. Based on the information contained in the PRECONDITION and SATISFACTION clauses of the procedure descriptions and the current state of the database, POISE will construct an appropriate series of procedure invocations to satisfy the user's goals.

7.0 CONCLUSION

Office tasks can vary widely in their complexity and problem-solving requirements. The POISE system supports tasks by automating actions that map directly onto office tools and by providing assistance to the office workers when problem-solving is required. By using a goal-based description of the tasks as well as a procedure-based description, POISE can support tasks that are carried out in nonstandard ways.

The current POISE system is a useful framework for investigating the problems involved with representing and supporting user tasks in office systems and user interfaces in general. However, it is difficult to evaluate the benefits of the POISE approach without actual user studies. It is hoped that by providing a high degree of flexibility in the POISE interface, the system can be configured to meet the needs of particular environments. The development of the POISE system has raised a number of interesting issues that either have not yet been addressed or are currently under investigation. The different methods of user procedure specification have already been mentioned. Another issue of central concern is the design of a knowledge representation that integrates the information required for all

aspects of the POISE system. An obvious example of the lack of integration in the current system is the separate knowledge bases for language analysis, language generation, and task recognition and planning. The representation of different user roles in distributed tasks is also being studied.

Acknowledgments

This research was supported in part by the Digital Equipment Corporation External Research Grant Program and by a grant from the Rome Air Development Center. The design of the POISE system was done in cooperation with Victor Lesser. Dan McCue, Norman Carver and Carol Broverman made major contributions to the implementation. Wendy Lehnert and David McDonald were responsible for the design of the natural language component of the system.

8.0 REFERENCES

1 Bailey, A.D.; Gerlach, J.H.; McAfee, R.P.; Whinston, A.B. "An OIS model for internal accounting control evaluation", *ACM Transactions on Office Information Systems, 1*, 1 (1983), 25-44.

2 Bates, P.C.; Wileden, J.C. "High-level debugging of distributed systems: the behavioral abstraction approach", *Journal of Systems and Software, 3*, 4 (1984), 255-264.

3 Barber, G. "Supporting organizational problem solving with a work station", *ACM Transactions on Office Information Systems, 1*, 1 (1983), 45-67.

4 Croft, W.B., Lefkowitz, L.S., Lesser, V.R., Huff, K.E. "Interpretation and Planning as a Basis for an Intelligent Interface", *Proceedings of the Conference on Artificial Intelligence*, Oakland University, Rochester, Michigan; 1983.

5 Doyle, J. "A Truth Maintenance System", *Artificial Intelligence, 12*, 3 (1979), 231-272.

6 Dyer, M.; Wolf, T.C. "McDYPAR – A Demon Parser", Department of Computer Science, UCLA, Los Angles, California; 1983.

7 Ellis, C.A.; Nutt, G.J. "Office information systems and computer science", *ACM Computing Surveys, 12*, 1 (1980), 27-60.

8 Ellis, C.A.; Bernal, M. "Officetalk-D: An experimental office information system", *Proceedings of the First ACM SIGOA Conference*, (1982), 131-140.

9 Fikes, R.E.; Henderson, D.A. "On supporting the use of procedures in office work", *First National Conference on Artificial Intelligence*, Stanford, California; 1980.

10 Gibbs, S.; Tsichritzis, D. "A data modeling approach for office information systems", *ACM Transactions on Office Information Systems, 1*, 4 (1983), 299-319.

11 Hammer, M.; Howe, W.G.; Kruskal, V.J.; Wladawsky, I. "A very high level programming language for data processing applications", *Communications of the ACM, 20*, 11 (1977), 832-840.

12 Hammer, M.; Zisman, M. "Design and implementation of office information systems", *Proceedings of the NYU Symposium on Automated Office Systems*; 1979.

13 Hammer, M.; Sirbu, M. "What is office automation?", *Proceedings of the First Office Automation Conference*, Atlanta, Georgia; 1980.

14 McCue, D.; Lesser, V.R. "Focusing and Constraint Management in Intelligent Interface Design", Technical Report 83-36, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts; 1983.

15 McDonald, D.D. "Natural language generation as a computational problem: an introduction", in *Computational Models of Discourse*, The MIT Press, Cambridge, Massachusetts; 1983.

16 Wright, M.; Fox, M.S. *SRL 1.5 User Manual*, Intelligent Systems Laboratory, Carnegie-Mellon University Robotics Institute; 1983.

17 Zisman, M.D. *Representation, specification and automation of office procedures*, Ph.D. Dissertation, Wharton School, University of Pennsylvania; 1977.

18 Zloof, M.M. "Office-by-Example: A business language that unifies data and word processing and electronic mail", *IBM Systems Journal*, *21*, 3 (1982), 272-304.