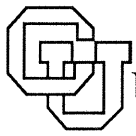
Using Workflow in Contemporary IS Applications

Gary J. Nutt

CU-CS-663-93 August 1993



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

USING WORKFLOW IN CONTEMPORARY IS APPLICATIONS

Gary J. Nutt†

CU-CS-663-93 August, 1993

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309-0430

† The author's telephone number is (303) 492-7581 and his electronic mail address is nutt@cs.colorado.edu. This work was supported by Najah Naffah, Bull S. A., Imaging and Office Solutions, 7 rue Ampère, 91343 Massy, France.

ABSTRACT

Using Workflow in Contemporary IS Applications

Contemporary businesses are facing a crisis in reorganizing their operation to manage their information processing needs. The evolution toward distributed systems has forced businesses to abandon centralized information system, and to try to adapt heterogeneous distributed systems to their extant information processing needs. Most existing information processing procedures inherently depend on centralized architectures, so the evolution is forcing the business to reorganize the fundamental way that it handles its information; this has come to be known as business process reengineering. Open systems have previously been identified as a key business and technology area to address business process reengineering; in this paper we describe how workflow technology can also be used as an important aspect of the solution for documenting, analyzing, and programming reengineered systems.

1. INTRODUCTION

Today's businesses face a major crisis in handling their information: traditional management information systems have been built using a mainframe machine that is controlled by large application programs working in concert with a local database. It has long been known that such configurations do not scale, and now the combined explosion of information and the hardware technology evolution have nearly eliminated pure mainframe computing. Multivendor distributed systems have replaced centralized computing, causing open systems to be identified as a key business and technology to address reorganization. Distributed systems, in turn have introduced the need for technology that will allow a group of users with their own workstations (or PCs) to collaborate on a single information processing job; software that supports this type of computing environment is generally referred to as *groupware* [9]. In this paper we describe how a particular class of groupware applications (workflow) can be used as a part of the solution for addressing the *business process reengineering* (BPR) problem.

BPR arises as a direct result of the hardware evolution: as centralized mainframe computing has been replaced by networks of client and server machines, the information processing procedures used by the business are no longer well matched to the computing environment. The distributed system can implement the same functionality as the centralized systems, but the business typically fails to take advantage of the full power and flexibility inherent in the distributed system when they use the original business procedures. As the information processing needs increase (in the face of an ever increasing pressure to be cost effective), businesses must make better use of the distributed hardware; this suggests that they must redesign the fundamental ways in which they manage information — BPR.

Open systems technology is relevant because of its focus on ways to allow hardware and software systems produced by competing vendors to be integrated into a single configuration [22, 27]. Workflow provides a simplified programming environment for implementing business procedures in a distributed computing environment.

1.1. What is Workflow?

Fundamentally, *workflow* is a technique for describing how a "procedure" can be accomplished by decomposing it into a set of discrete steps, then showing how information flows among them. The purpose of the procedure can vary from manufacturing processes to information management strategies. Variants of workflow have been used in many different disciplines to describe how procedures are intended to behave: PERT charts are a relative of workflow that describe how manufacturing and engineering processes can be organized to build a product. Queueing networks model workflow through a system of service providers in terms of service and interarrival times [15]. Flowcharts are another variant that describe how a sequential program should execute to process information.

A *workflow language* is a mechanism by which the steps are explicitly identified and the description of how work flows among the various steps is defined. The nature of the language for expressing the workflow approach depends on how the workflow specification is intended to be used:

(1) The language may be used exclusively to document a procedure so that it can be understood by everyone that has an interest in that procedure; this is a typical use of PERT charts.

(2) It may be used to specify the procedure's behavior under different loading conditions with varying service rates, allowing one to predict the performance of the procedure; this is how queueing networks are used.

(3) The language may be used to precisely define a set of actions that must be conducted for the procedure to work; this is typical of a flowchart-based visual programming language (and state diagrams).

Thus workflow representations range from informal, *descriptive* representations to formal, *prescriptive* representations.

PERT Charts. The primary purpose for a PERT chart is to pictorially represent the steps in a complex process, and to identify dependencies among the steps. PERT charts focus on the global pattern of *when* (in the process) tasks are to be accomplished, not on the details of any step. This general information about the procedure is easily interpreted by humans, even when they do not care exactly how any step is accomplished. (If a person is given a PERT chart representation of a procedure, it is expected that the person would have to determine how the steps are actually performed by consulting some other source.) The PERT chart is intended for human-to-human communication. The macro level view of the procedure is the focus of these procedures.

Refined PERT charts include time estimates for the expected length of time each step will take to execute; by adding this information to the chart, it begins to move from the domain of purely descriptive models into that of analytic models. After it has been annotated, the PERT chart can be used to determine a *critical path* through the chart that identifies which steps must be executed at the earliest possible time for the overall procedure to have the earliest completion time. That is, the descriptive model is sometimes refined into a simple analytic model.

Queueing Networks. Queueing networks are explicitly intended to analyze the performance of procedures and systems. The steps in the procedure represent generalized servers in the system, and the flow of jobs among the servers is analogous to the flow of work among steps. The details of steps do not define how a server accomplishes work, but rather, represent how long the server is expected to process a job as specified by a probability distribution function.

Flowchart Programming Languages. A flowchart is also a set of steps with an interrelationship among the steps that describe the order in which they must be executed to process specific work. The flowchart can be evolved into a computer program by refining each step until it is a prescriptive set of commands that can be understood by a computer, e.g., a basic block of source code. The final step specifications must be far more detailed, with less ambiguity and choice about how to accomplish the individual step, than those that appear in a PERT chart. That is, each step will ultimately have been refined to be an unambiguous command (e.g., a function routine or basic block of code) that can be interpreted by a computer. Both the view of the overall structure and the detailed view of each step in the structure are important for this use of workflow. However, since the steps are unambiguous specifications of action, it is not necessary for the overall intent (the *why*) of the procedure to be known to an "agent" that executes a step; such uses of workflow tend to eliminate information that describes the procedures intent or goals as spurious information.

Now consider the case where a human is to be delegated work in the context of a workflow procedure: either the procedure will have to be completely prescriptive — the flow and each step are completely and unambiguously specified — or it will have to accommodate the human's need to understand the context of the work and to be able to easily invoke other tools to obtain a concrete specification of the work or missing information to accomplish the work. That is, if a human is to execute individual steps of the workflow program, then the specification is likely to be much different for the human than it would be for the computer; the differences relate to level of detail (completeness of the step specification) provided, statement of goals and intent of the procedure, and the support of the human in invoking other tools while attempting to execute the step. This

issue has been a major barrier for successful workflow systems, e.g., see [25].

Workflow languages have an underlying formal model that define their syntax and semantics (i.e., their behavior); Petri nets [18] are one such formal model, and ICNs [6] are another. *Uninterpreted modeling languages* focus on describing the *flow* of work in the model, and ignore the details of processing within a step. The semantics for more detailed workflow models are represented by an *interpreted* formal model such as high level Petri (predicate/transition) nets [12, 14] or ICNs with activity interpretations. The amount and type of detail specification reflects on how the workflow language will be used.

1.2. Contemporary IS Requirements

IS systems are rapidly moving from centralized mainframe computing to heterogeneous distributed system, particularly using client-server computational paradigms. Today's client-server software is built so that application-specific parts of the program are executed on a client machine while application-independent parts are implemented in a remote server machine. For example, a server may implement a database while clients implement applications that produce reports from the database. Clients and servers intercommunicate using various levels of network protocols.

The proliferation of distributed systems technology is a severe blow to IS organizations that only have expertise in programming centralized mainframes. The first barrier for these organizations to overcome relates to assimilation of distributed systems technology so they can build effective distributed applications and groupware.

An enterprise can no longer obtain all of the software and hardware that it requires to solve its information processing needs from a single vendor. As mentioned earlier, this has led to the need for open systems, i.e., distributed systems designed to accommodate components supplied by many vendors. This has caused a revolution in the way computing equipment is acquired, and a secondary revolution in the way that IS organizations operate within their enterprise.

Distributed systems have created an even more severe crisis relating to the organization of information in the enterprise; the information age was born and reached a level of maturity under the centralized computing paradigm. It is now necessary for the enterprise to completely reevaluate how information is managed and used in the operation of its business. Those enterprises that ignore the problem will be left with archaic technology that will not allow them to compete; those that face the problem must derive an entirely new enterprise information architecture. The ramifications are that enterprises must now have a better understanding of information technology than was ever necessary in centralized mainframe situations, and they must also have better tools for designing and analyzing the enterprise architecture and computer systems that implement that architecture. The programming personnel must move to a new technology level to design and build contemporary distributed systems. *Workflow is one of only a few tools that explicitly address distributed systems and business process reengineering.*

The BPR problem in contemporary corporations cannot be completely addressed by technology, since many of the main issues are related to corporate culture, organizational theory, and so on [13, 29]. As in our earlier paper on office information systems [7], we focus on the technology as it is influenced by these other factors.

2. REPRESENTING PROCEDURES

Organizations operate on a set of basic principles and procedures. In some organizations there are extensive procedure manuals that describe how operations can be accomplished within the enterprise. Other organizations do not bother to write procedure manuals since it is difficult to express the procedures in a manner that can be understood by all of the employees. Secondly, the procedures tend to change as the

corporation gains knowledge about the procedures or as the operation of the business evolves. The problem is severely aggravated by business process reengineering. Third, the specification may be too prescriptive for human office workers to follow and still be effective.

Thus representing procedures is an important application for commercial workflow products. A good workflow system can provide a dynamic means for documenting the steps and their interrelationships for many procedures in the enterprise. (Not all work can be represented well with workflow; the work steps that cannot be represented may actually be more *principles* than procedures.) The workflow specification can be quite detailed if the enterprise intends that there be little deviation in the way that the procedure is to be executed (e.g., if the procedure must satisfy government regulations); conversely, it can be quite vague if the details of the steps are unknown a priori, or if the details of step implementation are really unimportant with respect to the overall organization. The level of representation reflects the concern of the procedure designer with how procedures are to be carried out in the enterprise.

The existence of computer technology can substantially change the way one represents office procedures. With the extensive realtime graphics facilities available, it is now possible to provide a broad spectrum of means to represent office procedures, e.g., see [8, 19]. Electronic representations have the advantage of being easily edited, updated, and distributed. This has always been a severe limitation of paper procedure manuals. Second, electronic representations can be interactive, allowing the user to navigate through the procedure specification using CAI-like technology. Third, electronic representations can incorporate animation facilities to provide more complete representations than are possible on paper. Fourth, electronic representations can incorporate multimedia, allowing the procedure to be described using text, graphics, images, and audio media. Thus, we view electronic descriptive workflow systems as an important aspect of workflow applications, albeit technically trivial compared to the other applications.

2.1. Descriptive Workflow Models

A workflow model that focuses on documenting and describing processes must provide facilities for identifying each step in the procedure, then showing how the execution of different steps are related. The model will be produced by humans (using computer tools) to communicate ideas to other humans. Therefore the modeling language and support tools are required to be flexible and concise in term of their ability to describe a procedure without being concerned with detailed descriptions of steps.

We have mentioned PERT charts as one language for describing procedures. In this section we introduce the *Information Control Net* (ICN) workflow model for describing applications.[1] (We first provide a skeletal definition that is sufficient for ICNs to describe procedures, then we elaborate on the definition in subsequent sections so that it can be used for analysis and programming.)

The steps in a procedure are called *activities* in an ICN. *Control nodes* are added to the language to specify that work may flow to (from) alternate activities when a predecessor activity has been executed — called disjunctive (exclusive OR) logic. ICNs also allow control flow to divide so that two or more activities can be executed concurrently (or in any order) — called conjunctive (AND) output control flow. Conjunctive input control flow is used to show that an activity can only be executed when all of its predecessor activities have completed — called AND input control. Pictorially, activities are represented by large, open circles, OR logic by small, open circles, and AND logic by small, filled circles. In Figure 1(a) we use an arc from activity A to activity B to indicate that activity A must complete before activity B can start, i.e., the work flows from activity A to activity B. Figures 1(b) and (c) illustrate disjunctive (OR) logic: in (b), the work will flow from activity A through control node B to either C or D, but not to both.

1. The description of ICNs that we provide in this paper is a substantial revision of the one presented in [6] or [7].

In (c) work from either A or B will flow through C to activity D. AND logic is illustrated in Figures 1(d) and (e): in (d), work flows from A to both C and D; in (e) activity D can only execute after work has flowed from both A and B via C. Ellis, et al., conducted extensive field studies using different variants of workflow models before selecting the particular model primitive shown in the figure [5]. It is interesting to note that these same operators are used to represent various patterns of control flow in parallel programming languages and systems, e.g., see Estrin, et al.'s GMB model [11].

The uninterpreted control flow can represent execution by showing how work flows through the net. This is accomplished by representing units of work as tokens that flow through the ICN graph, i.e., the net represents the steps and the flow rules, while a token can be placed on a node to represent that work is being done at that node. When the token passes through an AND node with multiple output arcs, the token is cloned and replicas are placed on each output arc; when a token is present on each input arc of an AND node with multiple input arcs, the node will fire, placing one token on its output arc. Thus token flow explicitly defines the way that work flows through the uninterpreted control flow net. In workflow terminology, the token is called a *workcase*.

The empirical experience with ICNs also suggested that the representation powers of the model could be much greater if the uninterpreted control flow net also represented how activities read and write various data repositories in the system. Let a square represent a data repository with an arc from a data repository to (from) a control node representing that the activity may read (write) data from (to) the data repository.

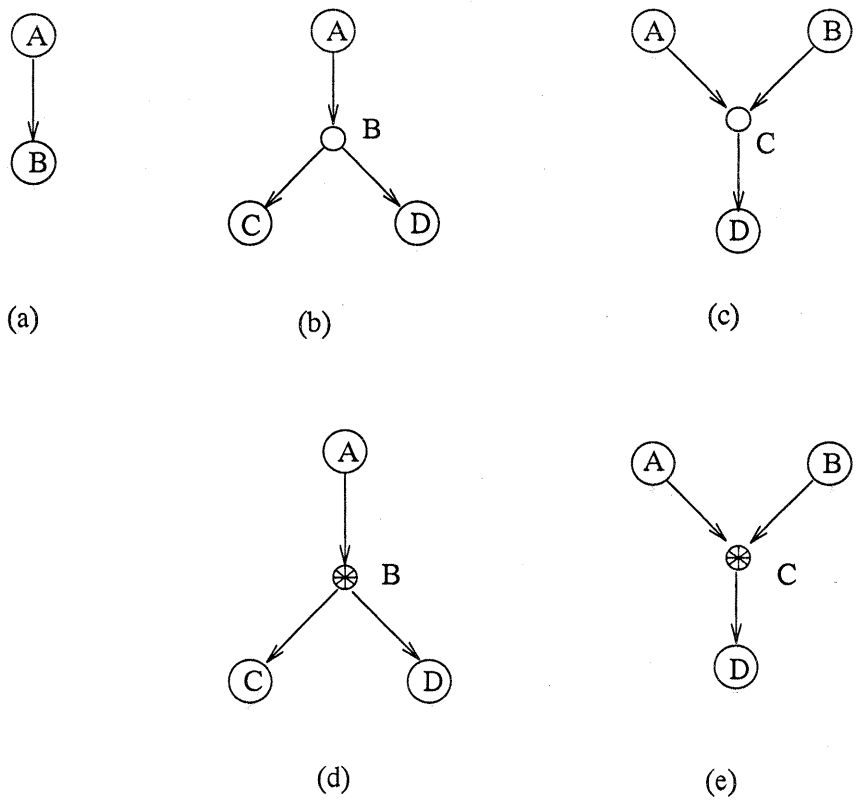


Figure 1: ICN Activities

In Figure 2, activity A writes information to repository E and activity D reads information from E.

2.2. Example: Purchasing Procedure

A typical purchasing procedure is for a person who needs a part to write a purchase requisition (PR) to cause the part to be ordered. If the person's supervisor approves of the purchase, then the PR is sent to the purchasing department so that a vendor can be selected and the part ordered using a purchase order (PO). On receiving the PO, the vendor supplies the part, packaged with a shipping list; the part is delivered to the purchaser's receiving department where the shipping list is compared with the shipment contents to verify that all parts on the order were received. The shipping department will deliver the part to the person that ordered it and notify the accounts payable department that it should pay the invoice for the part. Meanwhile, the vendor's accounts receivable department will have prepared an invoice and sent it to the purchaser's accounts payable department. When the purchaser's accounts payable department has received an invoice for a part and notification from the shipping department that the part was received, then it will issue payment to the vendor.

Figure 3 is an uninterpreted ICN to describe the purchasing procedure. Notice that even though there is no detailed explanation of any of the steps (activities and control nodes), the model represents how work flows through these two organizations (the purchasing organization and the vendor organization) to represent a business process. We use the uninterpreted ICN to represent parts of the external vendor environment as well as the procedure itself, since it is part of the description for how the purchasing procedure should work. We do not address exceptions in the descriptive model since the description is only the intended operation (handling "intended exceptions" can be added to the description if desired).

If we wanted to describe certain aspects of the process in more detail (without being "too prescriptive"), we could use a *nested* uninterpreted ICN to do so; e.g., Figure 4 represents a more detailed description of how to "select vendor." The vendor selection procedure may require that the purchasing agent talk to the purchaser to see if substitutions are acceptable. Then the purchasing agent selects a vendor and begins negotiation with him; this may be iterated on if negotiations between the purchasing agent and the vendor do not result in an agreement. Once agreement is reached, the vendor may actually begin preparing the part for shipment before receiving the PO, i.e., the PO number has not yet been issued. Even so, the work will flow to the next activity that will result in the preparation of the PO.

How can such models assist in business process reengineering or other procedure automation? The workflow model provides a concise, symbolic representation of the steps/activities that must be taken for

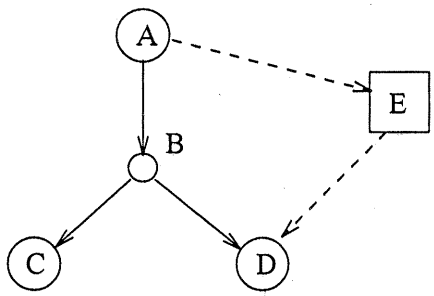


Figure 2: ICN Data Repositories

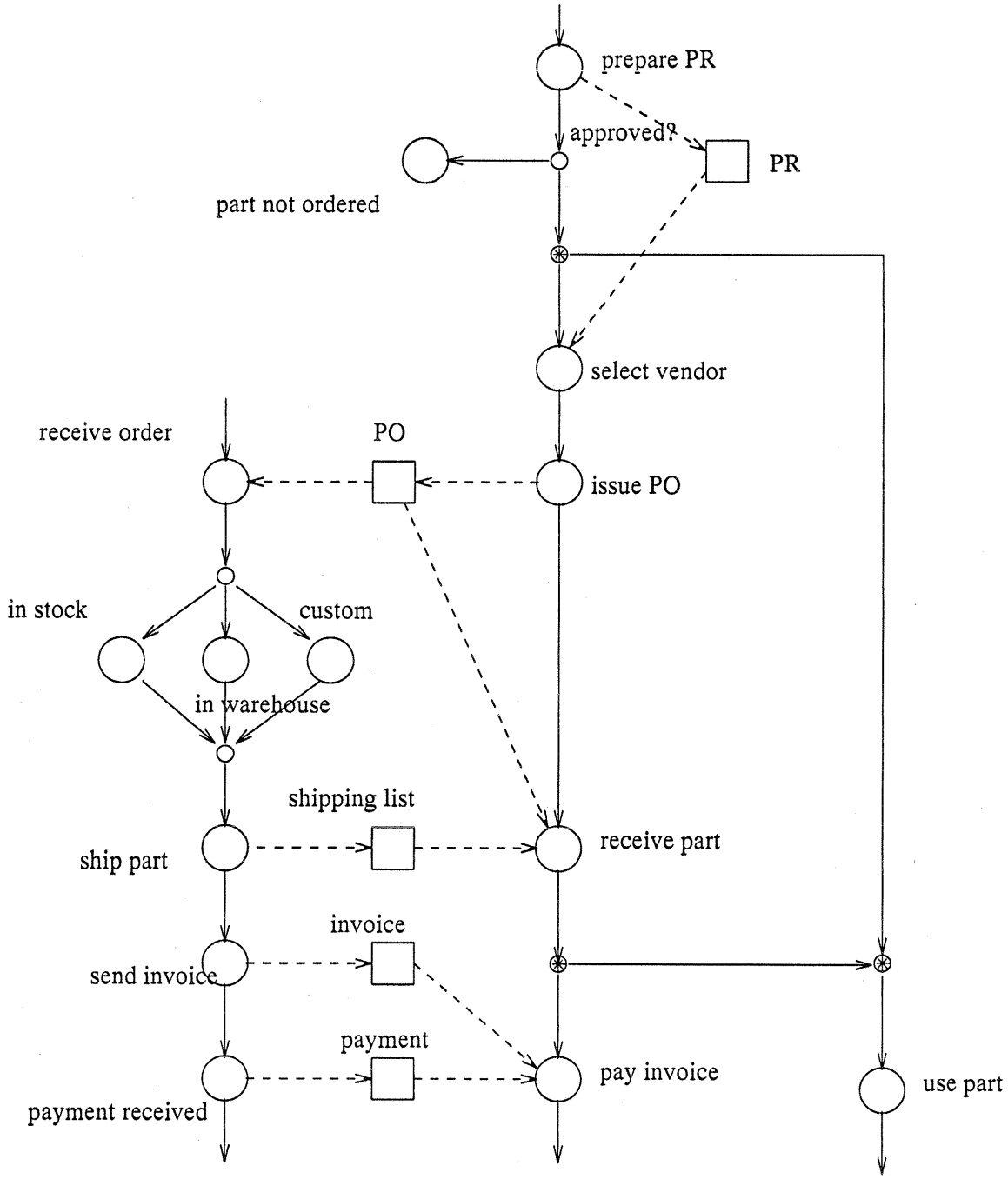


Figure 3: Descriptive ICN of a Purchasing Procedure

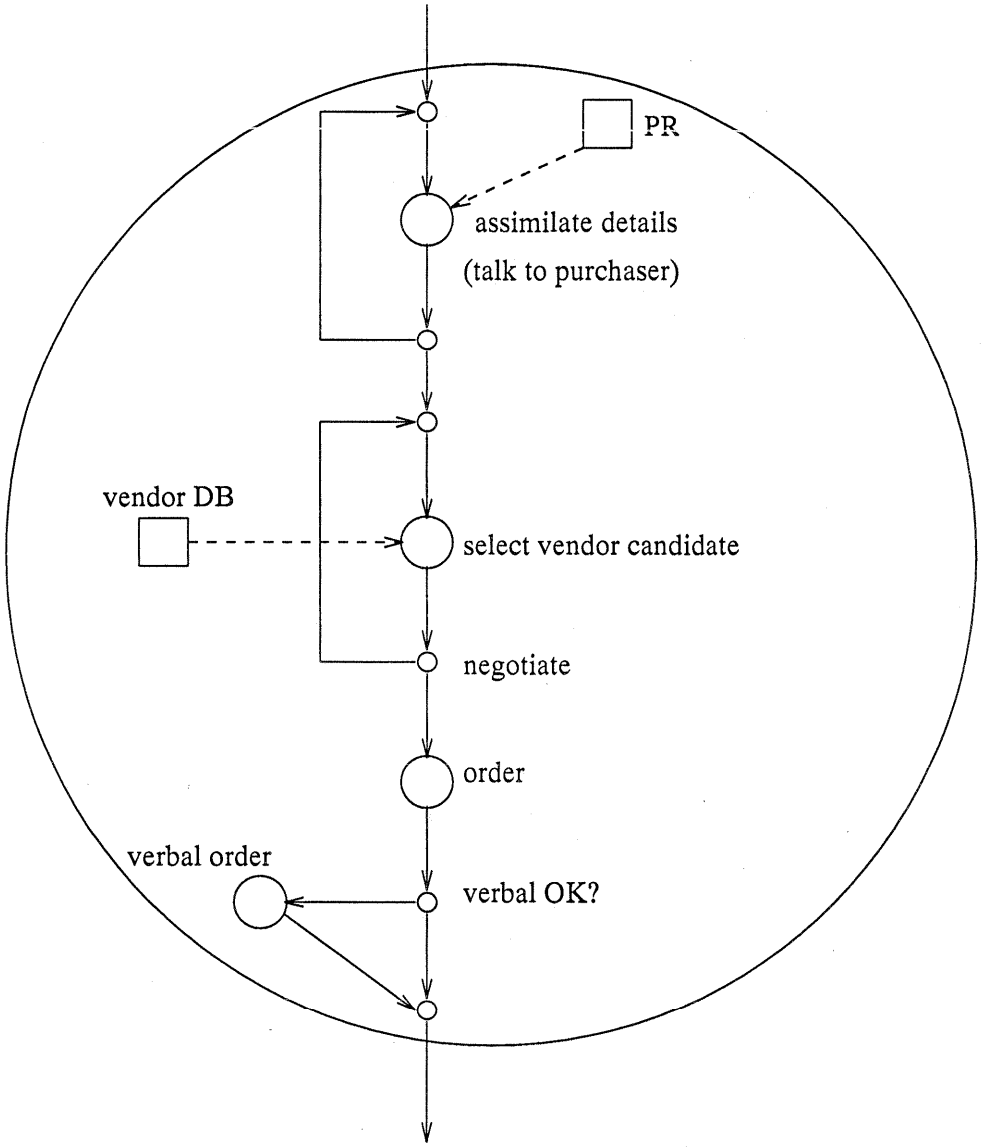


Figure 4: Nested ICN to Select a Vendor

the part to be ordered. Suppose that the policy of the purchaser's company is that verbal orders are not acceptable because the purchasing agent may not be able to get a PO issued prior to the time that the part is received in the receiving department? Or that if the vendor does not require a PO, then the purchaser will not issue one? The descriptive model can be used to allow or disallow each of these cases, depending if it is an important part of the policy being described by the ICN.

2.3. Actors and Roles

Basic ICNs model office work, independent of the assignment of the work to positions or individuals (including computers) in an organization. They do not explicitly take resource utilization into account; rather, the analyst is expected to model specific resources as they effect the procedure. This can be accomplished by explicitly describing when a specific agent executes an activity. A consequence of this

approach is that tokens may represent both workcases and agents (called *actors* in the workflow literature). While the approach is representationally complete, it has three significant negative aspects: it complicates the expression of the office procedure, second it makes it difficult to assign a single actor to multiple roles, and third it does not provide any obvious mechanism to represent actor scheduling/preemption (to multiplex across pending work). In another paper we discuss more formally how actors and their roles can be added to basic ICNs to produce extended ICNs [23]. In this paper, we use only an informal description of the concepts.

Actors may be people or computers, and their *roles* identify a set of activities that the actor is capable of performing in sequence (i.e, an actor represents one "processing entity," so it can only do one thing at a time). In extended ICNs, tokens represent only transactions (workcases) while all processing agents are represented with the role and actor constructs.

The role identifies a particular type of processing, e.g., a purchasing agent, but it does not identify the person or computer that can/will execute the role; that is done by explicitly *scheduling* an actor to a role for a workcase. Thus the model must associate (map) actors with roles, meaning that any actor is capable of playing the role to which it is associated. When a workcase needs to be processed, then an actor that is capable of playing the role is assigned to the workcase while it is in the activities associated with that role. It is also possible that any given human actor can hold many different roles (time multiplexing across those roles), such as purchasing agent, supervisor, employee, etc.; so this mapping is a many-to-many mapping.

Now we can provide a more elaborate description of the purchasing process by associating roles with the basic ICN activities (Figure 5). The purchaser role does many activities, including "prepare PR" and "use part", while the purchasing agent role is responsible for the activities "select vendor" and "issue PO". This added descriptive information allows one to provide more detail to the model without providing the details of the individual activities. It would also be possible to identify actors in the figure, but that would typically be of interest when the model is be analyzed or executed.

2.4. Descriptive Workflow Tools

A workflow tool that is to be used to support descriptive applications should have a formal underlying model, since it defines the conditions under which activities (or steps) are enabled for execution. Computer tools that support this application domain should include a graphics editor that has an ability to check models for syntactic correctness. (Such checks can be built into the editor itself, or provided as adjunct tools that can be invoke as required — each approach has its strong and weak points.) It must be easy and natural for business analysts to prepare and edit the models so they can quickly build and compare alternative business processes.

There are several tools that can be used to create and edit graphs; an important feature that distinguishes tools is the existence of the underlying formal model, including its applicability to the application domain. Tool differentiation often relates the quality and versatility of the user interface; as the products become more familiar to users, there will be increasing demand for CAI technology to support exploration.

3. ANALYSIS

Once workflow models are supported by computers, an obvious extension is for the system to also perform various analyses; therefore most computer supported workflow systems are also analytic workflow systems.

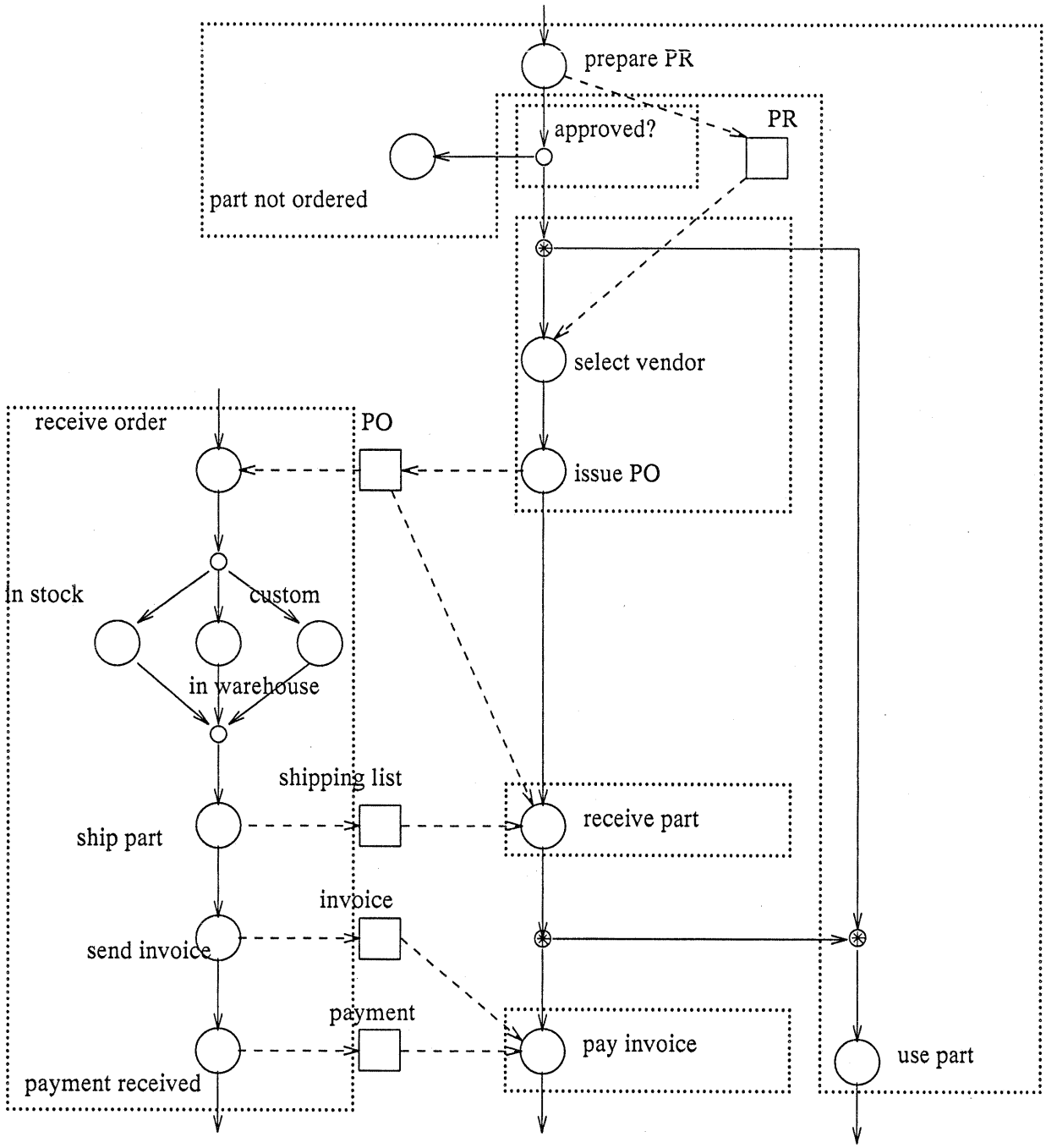


Figure 5: Actors and Roles in the Purchasing Procedure

The technical part of the BPR problem is heavily dependent on analysis tools. As the enterprise architect considers different alternatives, he needs to be able to describe the architecture in relatively precise terms, then to experiment with the model to understand its behavior under different loading conditions. Perhaps the simplest form of analytic support is to *animate* the token/workcase flow through the model; such a

tool does not necessarily provide quantitative measures of the performance, but it can give the analyst a valuable *qualitative* understanding of how the system will react to different loading conditions. For example, if the flow of the work through the system is subject to bottlenecks, then experimentation with the animation will often illuminate these areas with very little experimentation. Animation has become a fundamental "first cut" analysis tool for complex systems.

Qualitative observations of the performance of the system require more information in the model, more experimentation with the model, and more sophisticated support from the system. In some cases, the workflow system can be analyzed like a queueing network; this may result in closed form mathematical expressions to define performance measures. However, as the model grows in complexity, the time to analyze the model may grow rapidly, or it may simply become intractable, e.g., see [26]. We do not explore this approach in this paper, since the results from such analysis also typically require a strong background in statistics and mathematics before they are useful; it would be unusual for workflow designers to have this type of background. Instead, we focus our attention on qualitative analysis based on simulation.

Like animation, simulation requires that the model be interpreted. However, in simulation, the qualitative results are presented as post execution reports. Therefore, model interpretation need not be done in scaled realtime (in fact, scaled realtime is usually an annoyance when one wants the qualitative reports on system performance, e.g., see [24]). Such reports will normally address throughput, turnaround/response time, availability, and resource utilization. In a workflow model, resources are represented by actors, so the resource utilization statistics must correspond to the way actors do their work (within the role framework) under varying workcase loading conditions. The load on the system is workcases, so part of the reports will be related to characteristics of how quickly workcases get processed, how much delay they encounter at various scheduling points, etc. It is also worth noting that a workflow system probably cannot "automatically" produce the desired performance reports without some specification of *which* resources are of interest with respect to workcase flow; this suggests that the workflow system incorporates facilities for selective report generation (this concern is common in performance measurement domains, e.g., see [3]).

3.1. Analytic Workflow Models

The uninterpreted ICN model can be refined so that it is a model suitable for analysis. To accomplish this, one must add interpretations to the executable entities in the model to represent their individual performance characteristics. The complete workflow model will use these individual performance characteristics (e.g., a description of the amount of time a particular activity takes to execute on a particular type of workcase with a particular actor) to determine the overall performance characteristics of the model, then report the observations as specified by the analyst.

Roughly speaking, the performance behavior of a node (activity, control node, or data repository) is represented by specifying how long it should take for the activity to execute whenever the model semantics require execution. The simplest such characterization is to use an average execution time; the simulator moves tokens representing workcases from activity to activity as determined by the topography of the network, leaving the token on the node for a simulated time corresponding to the execution time characterization. The aggregate behavior of the system — actor utilization and workcase flow characteristics — is represented by the way tokens move through the graph model.

The model can have more sophisticated interpretations than mean value estimates of the execution time, including probability distribution functions, function subprograms, and verbal descriptions. If the model is to be used for simulation analysis of the system, the interpretation will normally be either a probability distribution function or a function subprogram.

Probability Distribution Functions. First, suppose that the interpretation is a probability distribution function: then the model can be used to represent a range and frequency of execution times for the activity, depending on the type of distribution used. For example, if the activity interpretation is a uniform distribution between 30 and 70, then any value between 30 and 70 is equally likely to be the execution time for any given execution of the activity; the first time the node executes, it might take 43 time units, and the next time it might take 67 time units. A simulator will typically support constant distributions, uniform distributions, normal distributions, negative exponential distributions, and erlang distributions; while understanding the properties of these various distributions requires that the analyst have a background in statistics, most analysts can make good use of the constant, uniform, normal ("bell-shaped") and negative exponential ("poisson") distributions to represent various patterns of execution time (most of these distributions are recognizable as standard shaped histograms or bar charts).

If we reconsider the purchasing example (Figure 5), then we might say that in the model, the time to prepare a PR is a constant time of 15 time units, the time to approve the PR is a uniform distribution between 10 and 100 (meaning that any particular workcase may require any number of time units between 10 and 100 with each number equally likely to be used), the time to select a vendor might be from a normal distribution with a mean value of 40 and a standard deviation of 5 (meaning that most of the executions will require between 35 and 45 time units, with decreasing numbers of executions occurring for smaller or larger values), etc.

These models are the basic fabric of queueing networks, thus a workflow model with probability distribution function node interpretations is essentially the same as a queueing network.

Function Subprogram Interpretations. Next suppose that the interpretation is allowed to be a software functions that returns a computed time. For generalization, let us assume that the function routine is passed data representing the workcase attributes when it is executed. For example suppose we have a C data structure of the form:

```
typedef struct {
        u_short      input_token;
        u_int        num_inputs;
        caddr_t      *data;
} USER_INPUT;
```

where the **input_token** represents a token identification, **num_inputs** represents an arc number from whence the token came (in the case that the node has OR logic and multiple input arcs), and **data** is a byte array containing the attributes associated with the token (workcase). Then the procedure can be written to read the characteristics of the workcase form the **data** array, and to compute the time to execute that workcase on the given activity. For example, the node interpretation may be of the form:

```
int activity_interp(USER_INPUT ptr)
{
// Parse the byte stream to get attributes
        parse(attr, ptr);
// Compute the execution time
        time = some function of the attributes;
        return(time);
}
```

Now the workflow simulator system moves tokens from activity-to-activity according to the logic rules specified by the graph, and calls the appropriate node interpretations (with the workcase description encoded in the input parameters) to determine the execution time. Thus one could easily write a function procedure to sample a histogram, or to compute an execution time based on arbitrary properties of the workcase (so this capability can easily implement the probability distribution function capability).

Workflow models such as the ICN model must incorporate a node to represent decisions, e.g., the "Approve?" node in Figure 5. How should the simulator choose which route the workcase should follow? The simplest solution is to attach probabilities to each output arc (such that the sum of the probabilities add to 100%). In the example one might say that the PR is approved 75% of the time and disapproved 25% of the time. Now, the simulator selects an output edge on which to route the workcase by sampling a uniform distribution between 1 and 100; if the sample is 25 or less, then it routes the workcase to the disapproved activity, otherwise it routes the workcase to the approved activity.

These stochastic models are easy to construct, but fail when the decision depends on the details of the workcase (since the arc selection is random). Again, the model can use function subprograms to select the outcome of decision nodes just as they were used to determine the execution time based on attributes of the workcase. In this case, the function routine is passed a pointer to the workcase, but it returns an arc label rather than an execution time:

```
int activity_out_arc(USER_INPUT ptr)
{
// Parse the byte stream to get attributes
        parse(attr, ptr);
// Select the arc
        arc_number = some function of the attributes;
        return(arc_number);
}
```

The simulator calls the **activity_out_arc** routine, obtains an arc identifier, then routes the workcases as specified.

Thus the **activity_out_arc** function for the "Approved?" node might obtain the cost attribute of the workcase representing the part being ordered, and decide not to order the part based on the current budget and the requester's job function (role).

An analytic model may also simulate the behavior of a data repository node. In the ICN model, the data repository may or may not be referenced by the activity interpretation, depending on the activity and workcase details, so the data repository interpretation is explicitly called from the activity interpretation

as desired. If an activity interpretation calls the **repository_access** function to read/write a repository, then the access is simulated by an analyst-defined routine for the repository.

For example, suppose that the "Select vendor" activity interpretation had the following schema:

```
int select_vendor_interp(USER_INPUT ptr)
{
// Parse the byte stream to get attributes
        parse(attr, ptr);
// Compute the execution time
        if(attr.amount < 100)
                time = uniform(5, 15);
        else
        {
                time = normal(50, 6) + repository_access(PR_ARC_NUMBER, ptr.data)
        };
        return(time);
}
```

In this interpretation, if the "amount" attribute of the workcase is less than 100, then the time is selected from a uniform distribution between 5 and 15, but if the amount is greater or equal to 100, the **repository_access** routine (also supplied by the analyst), is called to compute an execution time which is added to the sample from the normal distribution with a mean of 50 and a standard deviation of 6.

3.2. Analytic Workflow Tools

Analytic workflow tools are substantially more complex than descriptive workflow tools, since they must contain much of the functionality of the latter and additional analysis tools. Most tools are capable of animating and simulating workflow via a point-and-select interface, although the integrity of the underlying models is often suspect. Many systems provide the capability to specify probability distribution function interpretations, but far fewer allow the analyst to write function subprograms to define the behavior of a node or an output arc selector.

The Quinault ICN system [20] supported probability distribution function node interpretations for ICNs and the Olympus ICN system [21] supports both probability distribution function and C compatible function node interpretations for simulating workflow. These systems use basic ICNs to provide animation and simulation of the workflow (i.e., they do not model actors and roles).

Reporting facilities are an important aspect of the analytic tool; it must not generate too many reports on built-in parts of the model, since performance behavior of many parts of the model may be meaningless or out of the scope of interest of the analyst. We are unaware of any system (product or prototype) that has a general reporting mechanism, although Olympus provides some primitive customization tools.

As in descriptive workflow systems, the human-computer interface will be critical for initial acceptance. As the systems become better understood by the user, the modeling functionality will begin to dominate the tools success (since that is where the value of such a product ultimately lies).

4. PRESCRIBING A WORK PROCEDURE

We have explained how workflow addresses business process reengineering design with its representation and analysis aspects. Prescriptive workflow models are most concerned with *implementing* a groupware

system so that it conforms to the system that was designed and analyzed, and so that it takes advantage of distributed systems environments; i.e., a workflow system is also a tool to assist the programmer in designing and constructing distributed applications.

The representational power of workflow suggests that it is also an appropriate basis of a language for end users to program work within their own computing environment. There is empirical evidence that the ICN model fits the intuition of office workers when they describe a procedure; the same basic software development model used by the programmer should also be available to the end user in this context. The end user programming environment will differ from the programmer's environment in the nature of the language for interpretations and the available runtime facilities. End user programming will focus on integrating personal productivity tools and various forms of simple processing (although there is no reason that the full programming environment not be made available to end users).

Workflow implementations will generally be distributed application programs that support the information work of a group of people working on a common task, i.e., they will be to create groupware. (Of course as end user programming become prolific, parts of the system will be implemented for the use of an individual person.) Workflow systems encapsulate organizational knowledge in the model by explicitly describing the organization's procedures, and also by capturing the organizational hierarchy in the model.

It is important that the workflow system evolve through a specific analysis and implementation methodology — we believe that the methodology should be so complete that it uses *executable specifications* that are derived at analysis time and that evolve into a workflow implementation. (This conforms to many software engineering environments that use a comprehensive formal model for the requirements specification, design, and implementation phases of a product life cycle.) End user programming should be viewed as further refinement of the implementation in a manner that suits individual actors.

4.1. Workflow as a Programming Language

One goal of prescriptive workflow is to enable a group of persons to collaborate on the execution of a procedure. In establishing a collaborative framework, it is usually necessary for one person to assume responsibility for the overall goal, and to then organize and *delegate* work to other actors by defining a set of steps and the workcase flow among them (there are other social models for group work, but this one is the most widely used in business).

Workflow characterizes each of the parties involved in the collaborative work as being an actor. In some cases, a human actor delegates work to a computer actor in the workflow system; in other cases, the workflow system supports the delegation of work to other human actors. The acts of organization and delegation imply two distinct aspects: first a description of the work to be done must exist, and second responsibility for parts of the work must be handed off from one actor to another. The latter aspect suggests that there is a hierarchical authority among the actors, i.e., X cannot delegate work to Y unless X has the authority to do so.

The *delegator* is an actor that assigns work to another actor, and the *delegate* is an actor that is assigned work through the organizational hierarchy. Thus the delegator is a proactive agent in the system ("client") and the delegate is a reactive agent in the system ("server") for a particular delegation operation, but these roles may change for another operation.

When work is delegated, the delegate must be given the work in terms that are meaningful and acceptable. If the delegate actor is a computer, then the activity interpretation will necessarily be specific and unambiguous. If the delegate is a human, then the degree of prescriptiveness in the activity interpretations will vary according to the desires of the delegator (at the time the work is organized): highly

prescriptive interpretations imply that the delegator intends the way the work is carried out be tightly constrained. Less prescriptive interpretations suggest that the delegator want the activity to be completed, but does not care how it is actually accomplished; such interpretation must describe the goals, intent, and/or general instructions to the delegate.

Interpretation languages intended to be executed by computer delegates are classical procedural programming languages,[2] where the workflow graph determines the order in which the functions are called. The value of workflow technology in these applications is in *constructing* the workflow representation of the procedure by using the activities to modularize the work and the interpretations to specify the detail.

In the context of workflow, the graph and the interpretation languages are used to assist one human in describing a task to another human, i.e, the language should enable communication between the two humans, organize various computer tools to specify the task, and assist in its execution — be an instance of groupware. Croft and his associates have studied this aspect of the problem in detail, e.g. see [2, 16]; we are currently modifying our view of ICNs to better address this issue, e.g., see [10].

Any particular workflow procedure may be executed by a combination of human and computer actors — on an role-by-role basis. Thus, there is a recursive view of workflow in which a procedure may appear in an enterprise without any workflow context, i.e., it exists as a description/prescription for how some procedure can be accomplished should anyone decide they want to do that. Within the workflow procedure there is a specific control flow model to describe the conditions for an activity to be executed; however, the activity itself may be a nested procedure, definable with an ICN, or as some unit of work that a human actor will execute. In this paper we take a simplified view, and divide the discussion into parts that address procedure interpretations intended for computer execution and those for human execution.

4.2. Delegating Work to a Computer (Programming)

A large part of the value of using workflow models for writing distributed applications is that they provide a simple, visual language for organizing and defining a parallel computation, i.e., the power of this aspect of workflow is in *visual programming*, especially using the topdown methodology. Distributed business applications particularly benefit from workflow technology because of the intuitive use of visual models to represent concurrency in the information worker's domain of expertise.

The workflow system can be used for macro level design of procedures, e.g., describing how a PO is processed; for defining and implementing mid level computation, e.g., the steps in selecting a vendor to supply a particular part; and for micro level specifications, e.g., how to organize local information to fill in an invoice, and for binding work to a schedulable unit of computation (role). Professional programmers do not necessarily require visual, workflow-based languages to implement applications (although there is growing evidence that this class of language and analysis systems can be of substantial assistance in programming distributed systems [28]. There are several workflow languages, e.g., see [4, 17], although we continue our discussion of workflow concepts using the ICN language.

4.2.1. The Approach

In Section 3, ICNs were used to define analytic models by providing interpretations for nodes in the model. Prescriptive applications require more elaborate function subprogram interpretations and a comprehensive runtime system in which the interpretations can be executed. Rather than have interpretations return time values used for analysis, interpretations in prescriptive system focus on the computation

2. Workflow models are increasingly being cast as object oriented models in which activities are objects and workcases are represented by messages; this does not violate any of the assumptions made in this discussion.

that can be accomplished within the body of the function subprogram interpretation. As in analytic models, each activity and each control node can have a nested ICN or a function subprogram interpretation.

Nested ICNs. Prescriptive workflow applications can become very large programs, particularly as they represent full implementations of complex business procedures. This suggests the need for some mechanism to handle scaling and modularization of the model. This follows in part from the physical limitations of display screens, and in part from software engineering principles on modularization.

Since ICNs allow sequential control flow to appear in the graph or in node interpretations, there is no canonical form for an ICN; instead, the procedure designer can incorporate as little or as much disjunctive control flow in the graph/interpretation as taste dictates. However, as the ICN grows then either or both of the graph and individual interpretations may become arbitrarily large. The display size will tend to limit the size of the graph, even though the scale of the icons on the screen and scrolling mechanisms can allow surprisingly large numbers of nodes to be used in a graph.

Refinement can be implemented using traditional function procedures (within a textual interpretation), or using a nested ICN for any activity that has a single entry and a single exit arc.[3] Software engineering methodology suggests that function procedures should not be allowed to grow arbitrarily large (some leaders in the field suggest that a function should not contain more lines than will fit on a single page/display screen). Practice with ICNs suggest that a graph may be allowed to be larger than a screen, but not arbitrarily large; when a graph reaches some size threshold (dictated by the user), then nodes in the graph should be refined into nested ICNs such as those illustrated in Figures 3 and 4. Data are passed into (and received from) the nested ICN as attributes of the workcase tokens when they flow into (and out of) the ICN. This approach is called *visual abstraction.*

Function Subprogram Interpretations. A node is activated by the presence of one or more tokens flowing into the node. Since each token has attributes of the workcase that it represents, the workflow system must pass the attributes to the textual interpretation. Using the nomenclature from Section 3, an ICN node interpretation receives an input parameter of type **USER_INPUT** holding an encoded form of the token attributes. The interpretation does not return a value via the function name, so every interpretation (in the Olympus implementation) is of the form:

```
caddr_t activity_interp(USER_INPUT ptr)
{
// Arbitrary processing
        ...
// Olympus return
        task_return(u_short arc_number, caddr_t datap, u_int size);
}
```

3. The single entry/exit restriction is not strictly necessary, but it does simplify our explanation. It effectively restricts nested ICNs to activity nodes, suggesting that any nesting for a control node must be synthesized from an activity node.

The special procedure return is a call on the workflow runtime system to pass the output token attributes back to the workflow system so that they can be reassociated with a token. The **arc_number** selects an output arc for nodes with OR logic on the output arcs, causing a token containing **size** bytes of data (pointed to by **datap**) to be written to the arc. If the node has AND logic, copies of the token are placed on all output branches. Thus the Olympus implementation combines the node interpretation routine and the output arc selector routine described in the general discussion in Section 3.

The runtime environment must provide a variety of facilities to support activity interpretations. Below we discuss the most important aspects of the runtime environment. The system support includes graph manipulation and interpretation, workflow entity management, and scheduling. Other facilities are explicitly made available to a programmer via an application programming interface.

System Facilities. The runtime system implements the fundamental workflow interpretation algorithms described throughout this paper, including entity creation/destruction, storage allocation/deallocation, token movement, calling activity interpretations, system administration, etc. Many of these details are addressed as engineering problems at the time the runtime system is designed and built. Scheduling has some implementation-independent aspects.

Actors and roles imply several scheduling *mechanisms* that can be used to implement a variety of different scheduling *policies* as chosen by the enterprise. Mechanisms are defined by the ICN definition and runtime system, and polices are defined by the particular ICN model. Scheduling takes place at different granularities with different resources as summarized in Figure 6.

The ICN activity-role relation defines a schedule to assign activities to roles. While one may not expect that this schedule would change dramatically during execution of the procedure represented by the ICN, it is nevertheless possible.

The actor-role relation also defines a schedule which assigns actors to roles. It is easy to see many more circumstances in which the mapping of actors to roles varies with time, e.g., an employee takes a break so another temporarily assumes his role, or an employee is sick so some

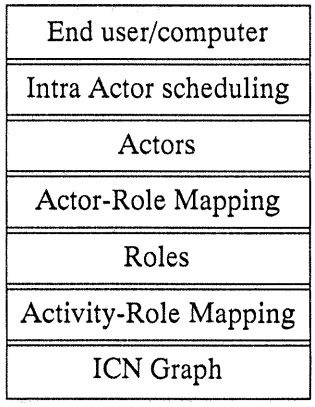


Figure 6: The Scheduling Hierarchy

other actor assumes his role(s) in the meantime.

In general, an actor is a sequential machine (a human or a process), therefore it can only execute one activity at a time. If there are multiple actors, then each of them can be executing an activity simultaneously so there is a notion of parallelism. If there are are two or more activities enabled for execution by a single actor, then the actor can only execute one of them at any given moment. This implies intra actor schedule is under the control of the actor.

The schedules and the marking determine the firing sequence for the ICN. When a token/workcase, w, enables a node, C_i, then the node is is said to be enabled to fire. However, all of the schedules provide additional constraints on the firing (execution) of the activity; that is, C_i can only be executed on w if it is assigned to a role, if an actor is assigned to the role, and if the actor chooses to execute the activity from its pending work queue. The workflow system handles enabling, activity-role scheduling, and actor-role scheduling, but the intra-actor scheduling is determined by the application and/or user.

The Application Programming Interface (API). The API defines the facilities provided by the workflow system for the use of the workflow programmer, i.e., it defines the runtime facilities available to a workflow programmer when he writes node interpretations. The API should support calls for data access, user interface management, and make provision for access to tools that are external to the workflow environment. End user programming support may either be a subset of the API or an entirely different API with a simplified conceptual model, depending on the design of the specific system.

Data References. ICNs distinguish between data access that is local to an activity and access that is shared across activities. (Local data access is defined by the programming language semantics in which the interpretation is written.) If data are to be shared across activities, then they can either be passed as token attributes (the dataflow approach), or explicitly stored to and retrieved from a data repository node. The technique for passing data as token attributes is described with the discussion of function subprogram interpretations. Repository access is accomplished using a generalization of the **repository_access** repository interpretation described in Section 3:

```
repository_access (u_short arc_number, caddr_t datap, u_int size);
```

In the ICN model, the logical equivalent of a token is received (or passed) from (or to) the data repository to represent read or write access — if the **arc_number** is from the activity to the repository, then the operation is a write operation, and if the arc is from a repository to an activity the operation is a read.

The data repository interpretation coincides with the **repository_access** interface — there must be a read routine and a write routine that is invoked when an activity interpretation calls **repository_access**:

```
caddr_t repository_interp(USER_INPUT ptr)
{
// Arbitrary processing
        ...
// Input return
        return();

// Output return
        return(caddr_t d_ptr);
}
```

The **repository_interp** schema is rudimentary, but defines the semantics that the model enforces on any data repository interpretation. Realistically, the **repository_interp** would be a part of the application programming interface to reference data stored in a globally accessible storage, e.g., a database.

The following program schema suggests how (in the absence of other support from an API) one might implement the activity entitled "Select vendor candidate" (from Figure 4):

```
caddr_t select_vendor_candidate_interp(USER_INPUT ptr)
{
        caddr_t r, s;

// Parse the data stream
        parse(attr, ptr);
// Build a query for vendors that provide the part
        q = malloc(...);
        build_query(q, attr.part, ...);
// Encode query
        r = malloc(...);
        encode(r, q);
// Query the database
        s = repository_access(VENDOR_DB, r, sizeof(r));
// Olympus return
        task_return(u_short arc_number, caddr_t s, u_int size);
}
```

The User Interface. Activity interpretations implement fragments of an application program that must interact with the end user. Therefore, the application programming interface must incorporate a human-user interaction model and appropriate facilities to implement the model. We do not address the details of the application programming interface in this paper.

System Administration. The workflow system must be administered to define actors, to establish actor-role mappings, etc. The details of system administration are implementation-dependent, thus we do not discuss them in this paper.

Using External Software. A workflow system exists in an IS organization in a general purpose

computing environment. Therefore the runtime system must make provision for the application programs and users to access and use external applications from within the workflow environment. The model only makes such a provision through activity function procedures; any procedure is an arbitrary program that may import and use an arbitrary API.

Support for End User Programming. End users will require a simplified activity programming model to assist them in defining macros, storing keystrokes, or defining other simple methods. They will initially require facilities to allow them to organize the use of their existing personal productivity tools, then later require full programming functionality.

For example, an end user may decide to delegate the procedure of filling out his time report, based on information kept in a personal calendar. Figure 7 is the ICN created by the end user; part (a) is the overall procedure, and part (b) is a nested ICN refinement of the "Clean up" activity. Figure 8 provides example interpretations for the activities in the graph; in the example we we only use the C programming language since we have not postulated any other language in this report. An implementation may provide a language more suited to end user programming, e.g., a 4GL language.

4.2.2. Problem Areas

Workflow models have historically had difficulty representing *unstructured* work in a procedure. This occurs when there is no detailed specification of the activities (or even among the activities) for some procedure, i.e., the activity will have to be executed by an actor that can reason from an incomplete specification — a human. The procedure can be defined in descriptive terms — goals and intent — but not in prescriptive terms.

Workflow *exceptions* are a special case of unstructured work; they occur when a workcase is being processed under a prescriptive workflow model, but in which the prescription is not complete enough to handle the case presented by the workcase-actor-activity combination. In the worst case, the workflow system does not detect that something is wrong, and blindly proceeds erroneously; in another case, a human actor detects an exception, but the workflow system does not allow the actor to correct the situation.

4.3. Delegating Work to a Human

When a workflow procedure is to be executed by a group of two or more humans, then the workflow model and runtime system serve the purpose of computer supported cooperative work (CSCW) systems, i.e., they assist humans in working together. The major distinction between a computer delegate and a human delegate is the reasoning ability possessed by the human. The computer delegate can only act on completely prescriptive descriptions of the work, whereas a human can be provided with imprecise ("partially prescriptive") descriptions of the work and still be expected to successfully execute the activity.

Thus the human is able to operate on activity interpretations with a broad range of prescriptive level. At one extreme, the human can execute the same interpretation that a computer can execute by following the interpretation without applying any reasoning or judgement to the job. At the other end of the spectrum, some humans can be told the goal and/or intent of the activity, and they will still be able to successfully execute the activity. As the interpretation becomes less-and-less prescriptive, the human delegate must be given more information about the intent of the activity *and* the human must have increasing amounts of domain knowledge to reason about the intent and to arrive at a successful execution.

The nature of an acceptable activity interpretation for delegating work depends on an agreement between the delegator and the delegate. In the absence of computer support, the delegator would tell the delegate

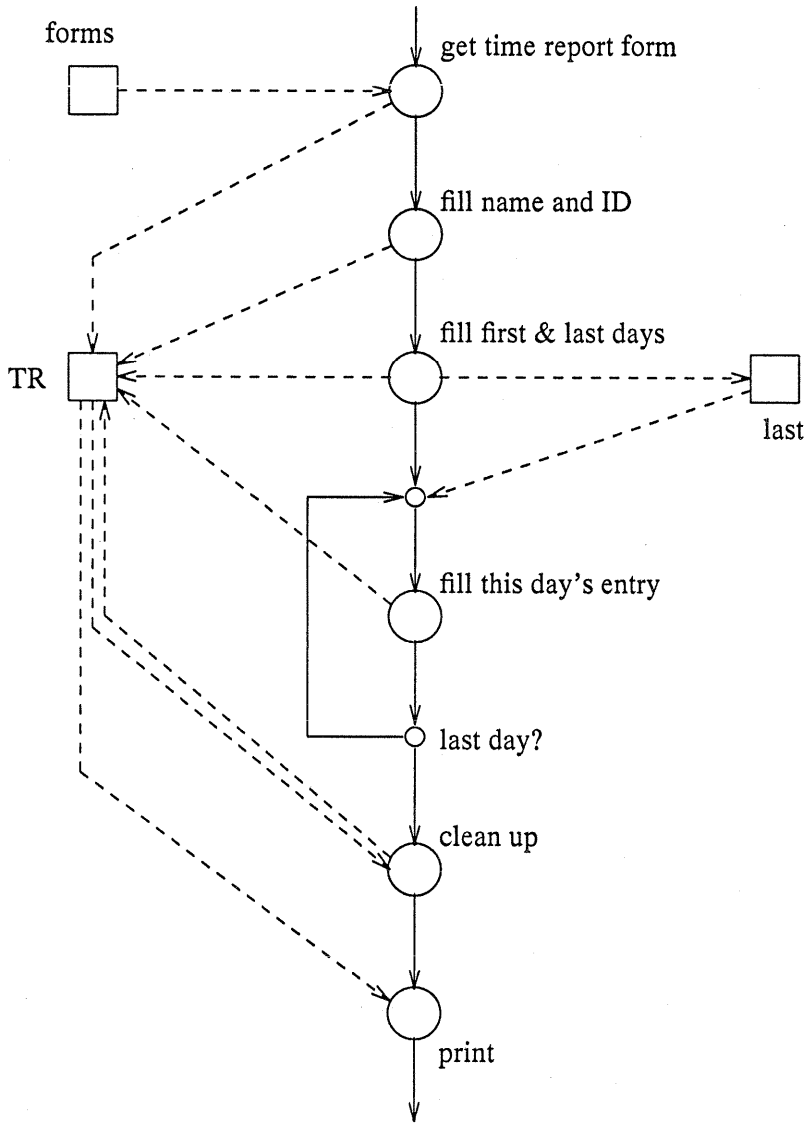


Figure 7(a): Time Report Procedure

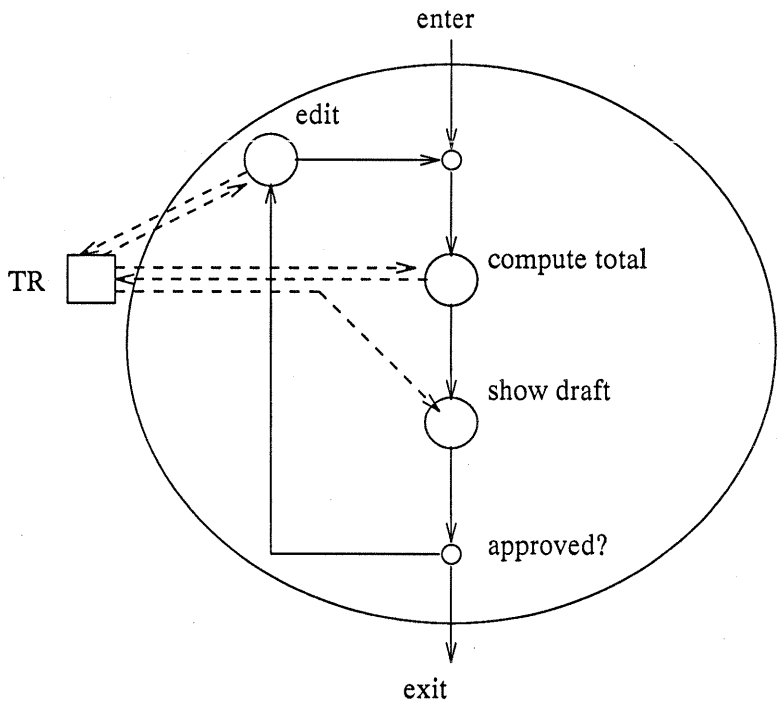


Figure 7(b): The "Clean up" Activity

```
caddr_t get_time_report(USER_INPUT p) {
        repository_access(READ_FORM, &q, sizeof(TR_FORM));
        repository_access(WRITE_TR, &q, sizeof(TR_FORM));
        task_return(NIL, NIL, 0);
}

caddr_t name (USER_INPUT p) {
        strcpy(m, "my name");
        strcpy(n, "my ID");
        repository_access(WRITE_TR, &m, sizeof(m));
        repository_access(WRITE_TR, &n, sizeof(n));
        task_return(NIL, NIL, 0);
}

caddr_t first_and_last (USER_INPUT p) {
        printf("Enter first day of period: ");
        sscanf(..., first);
        printf("Enter last day of period: ");
        sscanf(..., last);
        repository_access(WRITE_LAST, &last, sizeof(last));
        task_return(NIL, &first, sizeof(first));
}

caddr_t this_day (USER_INPUT p) {
        today = (int) p[0];
        value = today;
        repository_access(READ_CALENDAR, &value, sizeof(value));
        repository_access(WRITE_TR, &value, sizeof(value));
        today--;
        task_return(NIL, &today, sizeof(today));
}

caddr_t is_last_day (USER_INPUT p) {
        repository_access(READ_LAST, &last, sizeof(last));
        if(p.data == last)
                task_return(YES, &today, sizeof(today));
        else
                task_return(NO, NIL, 0);
}

caddr_t clean_up (USER_INPUT p) {
        API_icn(USER_INPUT p, rtn, ...);
        task_return(rtn.arc, rtn.data, rtn.size);
}

caddr_t print (USER_INPUT p) {
        API_Form_Print(TR, ...);
        task_return(NIL, NIL, 0);
}
```

Figure 8 (a): A Prescriptive ICN Interpretation

```
caddr_t fill_total (USER_INPUT p) {
        total = 0.0;
        do
                repository_access(READ_TR, &field, sizeof(field));
                total = total + field;
        until (field == NIL);
        task_return(NIL, NIL, 0);
}

caddr_t show_draft (USER_INPUT p) {
        API_Display_Form(TR);
        sscanf( ..., response);          // wait until through viewing
        task_return(NIL, &response, sizeof(response));
}

caddr_t is_approved (USER_INPUT p) {
        if(p.data == "y")
                task_return(YES, NIL, 0);
        else
                task_return(NO, NIL, 0);
}

caddr_t edit (USER_INPUT p) {
        API_Form_Edit(TR, ...);
        task_return(NIL, NIL, 0);
}
```

Figure 8 (b): The "Clean up" Activity

```
caddr_t TR_read (USER_INPUT p) {
        switch(num_input)
        {
        case FROM_PRINT: ...
        case FROM_SHOW_DRAFT: ...
        };
}

caddr_t TR_write (USER_INPUT p) {
        switch(num_input)
        {
        case FROM_PRINT: ...
        case FROM_SHOW_DRAFT: ...
        };
}

caddr_t calendar_read (USER_INPUT p)

caddr_t last_read (USER_INPUT p)

caddr_t last_write (USER_INPUT p)
```

Figure 8 (c): Data Repository Routines

what he is expected to do whenever the activity is to be executed, i.e., there will be a negotiation between the delegator and the delegate in which an agreement is reached about how work will be delegated (in

some cases, the negotiation is degenerate; the delegate is "trained" to execute a workcase with the delegator's standing interpretation). Even in cases where the interpretation prescribes the order in which steps must occur within the activity, the delegate actor may have to use extensive reasoning based on organizational knowledge.

The runtime system that supports the computer delegate provides a framework for supporting delegation to human actors. First, the workflow model establishes a context for the execution of the activity; the human delegate can usually use this information in reasoning about partially prescriptive interpretations. Second, the mechanism by which the runtime system delivers the workcase to the a computer delegate can be used to deliver the workcase information to the human delegate — however the delivery of the activity interpretation itself will differ in the two cases.

We currently have no concrete specification for the how goals and intent should be conveyed from the delegator to the delegate.[4] We observe that one possible language for expressing a nonprescriptive activity is to use a nested, descriptive ICN. This has the advantage of specifying workflow in cases where it matters, and allows the use of abstraction in cases where details do not matter.

Operationally, the user of a partially prescriptive workflow system should expect the computer system to provide tools to support them in routing/delivering workcases, to provide computer-based procedure descriptions, and to provide an integrated environment for using a spectrum of personal productivity tools.

4.4. Prescriptive Workflow Systems

Comprehensive workflow systems are characterized by the presence of a underlying formal model which can be used to capture knowledge of the organization in the structure of the model; in some cases they include other organizational knowledge relating to roles and organizational hierarchy among the roles. (We do not include systems that simply use electronic mail to transmit forms from one user to another; these products contain no organizational knowledge, but simply provide tools for moving electronic documents throughout the organization.)

Many products provide some variant of the computer delegate approach (without end user programming), but we are unaware of any that fully address the human delegate approach. The problem is sometimes addressed by simply embedding the computer interface within each activity interpretation; while this makes it possible to deliver workflow products in the absence of key technology, it has the danger of either forcing the API to essentially capture the essence of the human-computer interface, or of failing to support consistent interactions across different activities within a procedure or across procedures.

We believe that there are no systems that address the full spectrum of workflow application areas, and that systems to support prescriptive workflow for computer and human delegates is the area that lags the others. This is also the area where the descriptive and analytic modeling work can be of the most use, and where the payoff is the greatest using workflow technology.

5. DEVELOPING A WORKFLOW APPLICATION

A workflow system is explicitly designed to implement a policy for how a group of actors should interact. The model is defined at description and analysis time, then refined into an implementation as details are added to the model. One class of workflow systems takes the approach that the model is only descriptive, and that the operation of the workflow should be handled manually by the actors; an underlying electronic mail and forms system is sufficient to implement this approach.

---

4. However, Rick Blumenthal is studying a specific mechanism to assist in human-computer interaction in this context [1].

At the other end of the spectrum lie workflow systems that leave a minimum of flexibility for manual action to the actor. Such systems provide the most support and adhere most closely to the policies incorporated into the model; however, if an exception should occur in these highly prescriptive systems, then the user may have great difficulty in handling the exception "... because the computer won't let me do that ...". All activities must be highly prescriptive — even when the delegate does not care how the activity is completed.

The middle ground is a workflow system that allows certain aspects of an office procedure to be tightly specified and other aspects to be loosely specified. Tight specifications are appropriate when the procedure is designed to ensure that certains procedures are being strictly followed, e.g., due to government or contractual requirements. Loose specifications are appropriate when the procedure designer does not want to exactly prescribe how to accomplish some activity, but wants to give each actor an ability to do the job as they wish.

Workflow methodology is intended to take advantage of the topdown development methodology. It can be summarized as follows:

(1) Develop a descriptive model of the procedure. Iterate on the model until the procedure is an accurate description of the steps and the flow among them, even if the contents of some of the steps are vague.
(2) Annotate the descriptive with performance estimates to derive a simulation model of the procedure. The estimates for steps that are well-understood can be accurate, while the estimates for vague steps should use probability distributions with large deviations.
(3) Refine activities that are too vague for reasonable predictions by:
    - Writing function procedure interpretations
    - Defining a nested ICN for the activity
    - Identifying goals and intent of the activity
(4) Refine activities that are the leaf nodes on the hierarchy into function interpretations.

This technique follows topdown programming style using executable specifications at all stages of the development. The designer should iterate across steps as required when high level designs are inadequate for detailed designs that evolve with the development.

6. CONCLUSION

Business process reengineering requires clear thinking, careful analysis, and an agreement among the participants; the agreement may be arrived at through any form of negotiation, although we expect that the organizational hierarchy (who reports to whom) will have a significant effect regarding who are the delegators and the delegate vis-a-vis any particular unit of work. Descriptive and analytic workflow products are tools explicitly intended to assist in engineering an enterprise architecture. Therefore we see workflow as potentially being a major product to enable business process reengineering.

Prescriptive workflow models can be derived from descriptive and analytic models (as executable specifications) using a topdown development methodology. Further, the workflow model is natural for use in client-server distributed computing environments. As a programming tool, workflow has the benefit of visual programming for dealing with parallelism and the benefit of an straight-forward mapping into a distributed hardware system.

Workflow is a *tool* to assist in cooperative work — not a panacea. Clear thought and a cooperative organization are essential or any mechanism will fail in building distributed applications to support groups of information workers.

7. ACKNOWLEDGEMENTS

The author's understanding of workflow and its usage comes from many years of work with Clarence A. ("Skip") Ellis. This paper was written while the author was visiting Bull S. A., and was supported by Najah Naffah, Bull S. A., Imaging and Office Solutions, 7 rue Ampère, 91343 Massy, France.

8. REFERENCES

1. R. Blumenthal, *Interactions between Human agents and a Workflow System (tentative title)*, University of Colorado, Department of Computer Science, August, 1993.
2. C. Broverman and W. B. Croft, "Reasoning about Exceptions during Plan Execution Monitoring", *Proceedings of the AAAI-87*, 1987.
3. T. L. Casavant and J. A. Kohl, "The IMPROV Meta-Tool Design Methodology for Visualization of Parallel Programs", *Proceedings of MASCOTS '93*, San Diego, CA, January, 1993, 22-27.
4. E. Dyson, "Workflow", Release 1.0, EDventure Holdings, New York, September, 1992.
5. C. A. Ellis and P. A. Morris, "Information Control Nets: A Mathematical Model of Office Information Flow", *Performance Evaluation Review 8*, 3 (November, 1979).
6. C. A. Ellis, "Information Control Nets: A Mathematical Model of Office Information Flow", *Proceedings of 1979 ACM Conference on Simulation, Measurement and Modeling of Computer Systems*, August, 1979, 225-239.
7. C. A. Ellis and G. J. Nutt, "Office Information Systems and Computer Science", *ACM Computing Surveys 12*, 1 (March 1980), 27-60.
8. C. A. Ellis and N. Naffah, *Design of Office Information Systems*, Springer Verlag, Berlin, Heidelberg, New York, 1987.
9. C. A. Ellis, S. J. Gibbs and G. L. Rein, "Groupware: Some Issues and Experiences", *Communications of the ACM 34*, 1 (January 1991), 38-58.
10. C. A. Ellis and G. J. Nutt, "The Modeling and Analysis of Coordination Systems", *ACM 1992 Conference on Computer Supported Cooperative Work*, Toronto, Canada., November, 1992.
11. G. Estrin, R. S. Fenchel, R. R. Razouk and M. K. Vernon,, "SARA (System ARchitects Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems", *IEEE Transactions on Software Engineering SE-12*, 2 (February, 1986), 293-311.
12. H. J. Genrich, "Predicate/Transition Nets", in *Petri Nets: Control Models and Their Properties, Advances in Petri Nets 1986, Part 1*, W. Brauer, W. Reisig and G. Rozenberg (editor), Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg, New York, 1987.
13. V. Gurbaxani and S. Whang, "The Impact of Information Systems on Organizations and Markets", *Communications of the ACM 34*, 1 (January, 1991), 59-73.
14. K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Springer Verlag, Berlin, Heidelberg, New York, 1992.
15. E. D. Lazowska, J. Zahorjan, G. S. Graham and K. C. Sevcik, *Quantitative System Performance*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1984.
16. L. S. Lefkowitz and W. B. Croft, "Planning and Execution of Tasks in Cooperative Work Environments", *IEEE AI*, 1989.
17. R. T. Marshak, *Workgroup Computing Report*, Patricia Seybold Group, May, 1993.
18. T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE 77*, 4 (April, 1989), 541-580..
19. W. M. Newman, *Designing Integrated Systems for the Office Environment*, McGraw-Hill Book Company, New York, NY, 1987.

20. G. J. Nutt and P. A. Ricci, ''Quinault: An Office Environment Simulator'', *IEEE Computer 14*, 5 (May 1981), 41-57.

21. G. J. Nutt, A. Beguelin, I. Demeure, S. Elliott, J. McWhirter and B. Sanders, ''Olympus: An Interactive Simulation System'', *1989 Winter Simulation Conference Proceedings*, Washington, D. C., December 1989, 601-611.

22. G. J. Nutt, *Open Systems*, Prentice Hall, Englewood Cliffs, NJ, 1992.

23. G. J. Nutt and C. A. Ellis, ''Adding Actors and Roles to the Basic ICN Model'', University of Colorado Department of Computer Science Technical Report in preparation, August, 1993.

24. C. J. C. Schauble, ''A Memory Access Simulator for MIMD Machines'', University of Colorado, Department of Computer Science, PhD proposal, April 1989.

25. L. A. Suchman, ''Office Procedures as Practical Action: Models of Work and System Design'', *ACM Transactions on Office Information Systems 1*, 4 (October, 1983), 320-328.

26. M. K. Vernon, J. Zahorjan and E. D. Lazowska, ''A Comparison of Performance Petri Nets and Queueing Network Models'', Technical Report #669, Computer Sciences Department - University of Wisconsin, Madison, September 1986.

27. T. Wheeler, *Open Systems Handbook*, Bantam Books, New York, NY, 1992.

28. *IEEE Software 8*, 5 (September, 1991).

29. *Communications of the ACM 34*, 12 (December, 1991).