The evolution toward flexible workflow systems

Gary J. Nutt †
Department of Computer Science, CB 430
University of Colorado
Boulder, CO 80309-0430 USA

Abstract. The simultaneous evolution of personal computing tools and networks has focused attention on the notion of harnessing computer technology to assist in human collaboration on group work. While personal productivity tool technology and use have reached a high level of sophistication, the most basic ideas for how computer technology should assist in collaboration across the network have not yet converged. The approaches range from ones where coordination of work is uniquely human-controlled, to workflow-based approaches where the computer is involved in scheduling the group's work. This survey paper describes how workflow technology has evolved from a modeling focus to flexible model-based systems to support collaborative work across this range of work styles.

1. Introduction

In the last 15 years, personal computing has grown so powerful it has fundamentally changed the way people conduct business. However, this new technology has had difficulty providing well-accepted ways of supporting collaborative computing among groups of people. In particular, the technology has not exploited distributed systems to a significant degree. Successful computer tools supporting interactions among individuals using computers are generally limited to sharing a common information base (e.g, a database, bulletin board, news group, or web page), and the exchange of information using the telephone, electronic mail, and FAX.

1.1. Two Approaches to Supporting Collaborative Work

In general, *groupware* is software that attempts to use computers and networks to provide services to assist a group of workers in conducting their work — hence the closely-related term *computer-supported cooperative work* (CSCW) [1, 2, 3]. While there is little argument concerning the need for computers to provide some kind of support

† This work was supported by NSF Grant No. IRI-9307619

for collaborative work, within the CSCW research community there is considerable difference of opinion about *how* this might best be accomplished. The two basic approaches are called situated work and workflow.

The *situated work* camp advocates the use of evolutionary systems to provide increasingly sophisticated personal productivity tools, with little explicit attempt to have the system stage the work to be performed. The target computing environment contains a desktop with a tool repertoire. The human participants decide how tools should be applied to perform the work, and how work should be routed from one person to another.

The *workflow* camp advocates the use of models and systems to define the way the organization performs work. In particular, a work *procedure* in a workflow system is defined by a *workflow model* composed of a set of discrete work *steps* with explicit specifications of how a unit of work flows among the different steps (e.g., see [4, 5, 6, 7, 8, 9, 10, 11]). Whereas the situated work approach explicitly omits step definition and inter-step coordination, workflow explicitly includes it. Besides assisting in the coordination and execution of a procedure, workflow models have also been used to *describe* how the procedure performs the work, and to *analyze* the behavior of a procedure.

Over time, neither situated work nor workflow has dominated the way general CSCW systems are designed. There are strong arguments for why one approach or the other is appropriate in different situations; in [12] Suchman carefully explains how a prescriptive workflow system would have simply complicated a group's work for the case described in her classic paper; the feature articles in [13] update that view and explain the difficulty in representing how work should be performed. Robinson and Bannon argue that work models can never truly represent an organization's work because of ontological drift and other factors [14]. On the other hand, Malone and Crowston highlight the importance of coordination and computer support for coordination in contemporary organizations [15].

This leads us to the idea that the contribution of computer technology might best be achieved by taking a perspective in which the system can be used to support both situated work and workflow (or some customized combination of the two) on a case-by-case basis. Therefore, in workflow technology, we see an evolution toward flexible workflow models and systems that provide robust means for supporting situated work in a distributed environment.

1.2. The Evolution of Workflow

The term "workflow" began to be widely used in the mid 1980s. The technology evolved from work in the 1970s on office information systems. In 1977, Hammer et

al., described their experience with a very high level application language, BDL, to route forms among different processing stations in a company [16]. At about the same time, Zisman was finishing his thesis at the Wharton School, defining a language for representing and analyzing office procedures [17]. In 1980, Ellis and Nutt described the needs of office information systems to the computer science committee, particularly addressing models of office procedures [18]. In the early 1980s, interest continued to build in office information systems, including workflow technology, culminating in the introduction of the ACM Transactions on Office Information Systems, and the creation of the annual Conference on Computer Supported Collaborative Work.

The technological hopes, schemes, and promises for "office automation" met the realities of the requirements of industrial-strength products by the mid 1980s. Workflow did not meet the expectations of business users, and began to be roundly criticized as a solution for supporting collaborative work, e.g., see [19]. Today, workflow researchers have come to realize that the technology must address issues that span anthropology, ethnography, organizational psychology, sociology of office work, as well as computer science technology [1].

This paper is a *survey of workflow technology* (not a survey of commercial workflow systems such as can be found in [5, 9, 20, 21]). It describes an evolution in which technology has relaxed its strict coordination approach. In the last part of the paper, we show how this evolution will continue so that the next generation of workflow systems can be designed to fit naturally into modern organizations of people.

We organize our survey to correspond with the way the technology evolved in the area. As a pedagogical tool, we introduce a work procedure example in the next subsection. In Section 2 we focus on languages and systems for modeling collaborative work as workflow. In Section 3 we identify the additional facilities an enactment system must provide over modeling systems. Section 4 discusses workflow enactment languages and their generalizations, then Section 5 complements this discussion with remarks regarding generalized workflow management systems.

1.3. An Office Procedure

There are many different kinds of procedures used to describe the way work is done in an organization. Some of the procedures are inherently vague (since no individual really understands how the work is accomplished), while others are highly-refined and highly-specified to ensure rigid and predictable execution of the procedure. We use an example that has small cases to represent a spectrum of procedure types. There is an expanded discussion of the example in [22].

In most corporations, government agencies, and universities, there is a standard procedure by which an employee can purchase goods from an external vendor. The

employee fills-out a *purchase requisition* (also informally called a "PR"), then has it approved by his or her supervisor. Once the PR has been approved, it is sent to the purchasing department where it will eventually be processed by a purchasing agent. This agent reads the PR then selects a vendor, based on the requirements stated on the purchase requisition, standing rules regarding purchases of this type, and the result of negotiations with vendors. (Very often, the vendor selection process is guided by existing contracts with the vendor, but also it is often driven by informal knowledge the purchasing agent has about different vendors, e.g, their delivery record, ability to provide service, the quality of their products, how this transaction will relate to other transactions, etc.)

Once the vendor has been selected, the purchasing agent issues a *purchase order* ("PO") authorizing the vendor to ship the goods to the company. One copy of the PO is sent to the vendor, another to the receiving department, another to the accounts payable department, and a final copy is retained by the purchasing agent. When the goods are delivered to the receiving department with a *shipping list*, they are correlated with the PO; any discrepancies between the order and the receipt (including backordering, discontinued goods, etc.) are noted and sent to the accounts payable and purchasing departments. Meanwhile, the vendor issues an invoice to the accounts payable department requesting payment for the goods on the PO that were actually shipped. The accounts payable department correlates the invoice, the PO, and the shipping list, and if everything is consistent, issues payment for the goods that were received.

In an abstract sense, this group procedure is easy to understand. Because of the details in the procedure — possibilities of several different outcomes from individual parts of the procedure — computer support for the work can be very difficult to achieve. We discuss these details throughout the remainder of the paper.

2. Models of Procedures

A *model* is an abstract representation of a target phenomenon: a model airplane is an abstraction of a real airplane that represents the (scaled down) appearance of the real airplane. An economic model represents how money, services, goods, etc. interact, in some target society. A PERT chart represents the order of steps in product design and manufacturing. A model represents a subset of the characteristics of the target system, i.e., it uses specific characteristics of the target system while ignoring others. A workflow model represents a group work procedure; like other models, it explicitly captures certain characteristics of the target while ignoring others.

Which characteristics should appear in the workflow model, and which should be ignored? That depends on the purpose of the model. A model of a procedure can be used

to describe a procedure to group members (*representation*), for predicting the behavior of the procedure's execution (*analysis*), and as a language for capturing and comparing notions regarding the procedure's implementation (*design*). A model of the purchasing procedure in Section 1 intended for teaching new employees how the procedure works might focus on describing how authorizations are obtained, how people in purchasing agents interact with vendors, etc. If the model were used to analyze the staffing requirements of the purchasing department, characteristics such as the distribution of purchase requests, the amount of time required to identify a vendor, etc. would be critical. If the model were to be used to design a procedure to take advantage of an information infrastructure, the characteristics of the model might include information flow patterns, computer connectivity, storage location and capacity, etc.

The utility of a workflow modeling language lies in the match of the language constructs to the domain in which it will be used. A successful language must have facilities by which the modeler can easily represent, analyze, or design procedures. If an instance of the language — a particular model of a procedure — is easy to understand, it is potentially useful for representation. If the instance easily accommodates many kinds of metrics, it has the potential for being useful for analysis. If it easily represents essential aspects of the way a procedure is to be defined, it is potentially a useful model design language.

2.1. Workflow Modeling Language

There is an extensive history in discrete system modeling language development, ranging from computer simulation languages to iconic visual languages [23, 24, 25]. Many of these languages, including workflow modeling languages, adopt a *data/control flow* perspective. A workflow model represents how a unit of work (also called a *workcase* or *job*) flows from one processing step (also called an *activity* or *task*) to another. Thus, a workflow language contains constructs to define a set of steps to represent units of work, a sequential *computation language* to provide an interpretation for each step, and a *coordination language* to define how the work flows among the steps [4, 5, 6, 7, 8, 10, 11, 26].

In workflow modeling languages, the modeling domain determines the nature of the computation language, though the same coordination language can be used across different modeling domains.

2.1.1. Computation Languages for Modeling The fundamental purpose of a representation computation language is to enable one person to describe how a step is accomplished for the benefit of other human readers. This type of computation language concentrates on features that can be used to provide clear and intuitive explanations,

e.g., the language may be a set of hypertext explanations for each step [27]. In the purchase order processing example, a step such as "select a vendor" could be defined by an explanation of the factors the agent uses in choosing the vendor for a particular part.

An analysis computation language must allow the user to quantify various aspects of a step's execution, e.g., it may provide tools to specify arrival and service time distributions, a way to stochastically choose from among a set of alternative control flow paths, and a way to instrument executable (simulation) models [28, 29, 30]. In the purchase order processing example, a step such as "select a vendor" might have a service time distribution to represent the amount of time it takes to select a vendor.

Design languages are widely-used in software engineering to capture requirements, constraints, relationships, and algorithms for implementing an individual software component. In environments supporting executable specifications, the design language can also be the implementation language [31]. In the purchase order processing example, a step such as "select a vendor" might have an architecture for how a database is used to screen vendors, how the agent reviews a vendor's offerings using a web page, etc.

2.1.2. Coordination Languages The coordination language specifies how a workcase passes from step-to-step [5, 8]; this is an identifying characteristic of workflow languages. Coordination languages can use a spectrum of techniques to accomplish their task, ranging from rule-based approaches (e.g., see [32, 33]) to explicit precedence (see below). In his thesis, Zisman combined the two to define the coordination scheme [17]. Since precedence-based coordination languages dominate in contemporary workflow systems, we base our discussion on this approach.

For workflow, each step — such as "select a vendor" — is represented by a node in a graph; execution precedence is established defining edges between the nodes. Suppose x, y, and z are nodes (corresponding to steps in the workflow specification); if the graph includes a directed edge from x to y, (x, y), the work defined by the computation language instance for the step corresponding to x must be completed before the computation for the step corresponding to node y can be started. This type of coordination is common in group procedures such as the purchase order procedure, e.g., we could model part of the procedure by letting x be a step representing "select a vendor" and having y represent a step to "issue a purchase order". This means that in our procedure, we must select a vendor before issuing a purchase order.

Traditional precedence graphs are not expressive enough to easily represent all the coordination paradigms that normally arise in office procedures. Therefore, the coordination language is embellished to explicitly represent disjunctive and conjunctive control flow [34].

Disjunctive ("exclusive-OR") control flow represents the situation where a workcase

may flow from step x to step y or to another step z (but not both). For example, if a step, x, represents "approval of a purchase requisition," then the next step in the workcase might be step y ("request is approved"), or it might be step z ("request is not approved"), depending on the outcome of the approval step.

Conjunctive ("AND") control flow represents the situation where both steps y and z are to be executed after x has completed. This situation also occurs in many procedures: a workcase may be "split" for detailed processing by two different parts of the procedure concurrently. For example, a purchase order may be split into two parts in step x ("distribute the purchase order") so that the receiving department recognizes incoming goods related to the purchase order in step y (and its successors), and the accounts payable department recognizes the corresponding incoming invoice in step z (and its successors). Following the conventional use of the term "concurrency" that is associated with the AND control flow in distributed software, the y and z parts of the procedure might be executed simultaneously or in either order.

A Petri net [35, 36] is a control flow model that explicitly represents sequence, OR-flow, and AND-flow, thus Petri nets (or their logical equivalent, again see [34]) are often the underlying coordination model in a workflow language.

In general, a workflow coordination model can be defined as a *directed graph*, (N, E), with a node set N representing individual steps in the procedure and an edge set E representing the coordination structure among the tasks. In the case of Petri net coordination models, the edge set is defined as a set of *multi-edges*; any edge $(x, y) \in E$ can have multiple nodes at its head or tail. If x represents a set of nodes {r, s, t}, then the multi-edge has task nodes r, s, and t at its tail and node y at its head. If y is a set of nodes {u, v, w}, then there is said to be a multi-edge from x to nodes u, v, and w. Multi-edges are used to represent conjunctive control flow. Of course E can also contain simple edges in which x and y are individual task nodes; combinations of simple edges represent disjunctive control flow.

The Petri net coordination model represents partial orders of step execution; if a procedure has been decomposed into steps $n_1, n_2, ..., n_k$; then a coordination model allows one to describe any partial order on the execution of the steps for processing a workcase. If $(n_i, n_j) \in E$, then n_j cannot begin executing until after n_i has terminated, e.g., n_i might be a step where the vendor logs in a purchase order, and n_j might be a step in which items in the order are checked to see if they are in stock.

More complex situations can also be specified using multi-edges and multiple arcs; e.g., if n_i can be executed when n_j and n_k or just n_m has completed execution, then the multi-edge $(\{n_j, n_k\}, n_i)$ and the simple edge (n_m, n_i) are both included in E, i.e., there is a multi-edge from n_j and n_k to n_i and a simple edge from n_m to n_i. In the purchasing example, this scenario occurs if an accountant considers paying an invoice (n_i) when either a purchase order has arrived from purchasing (n_j) *and* a shipping list has arrived

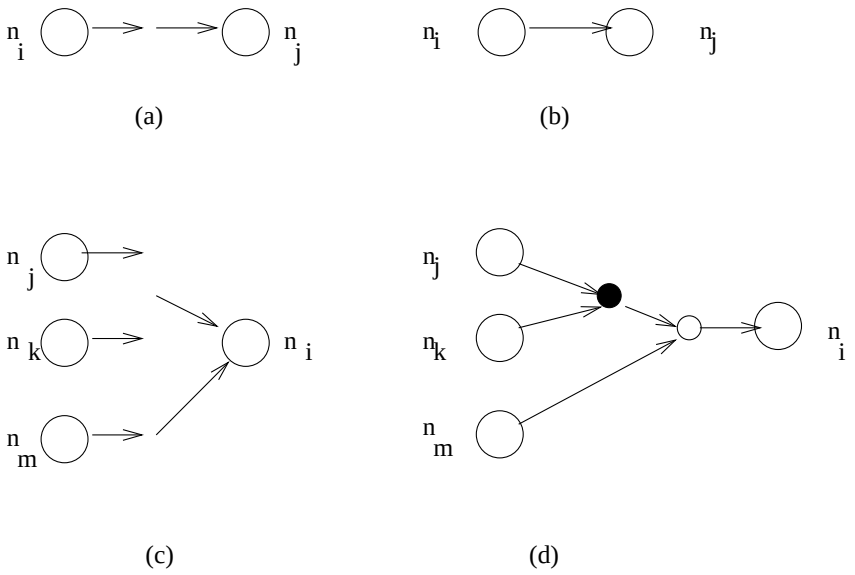

Figure 1. A Visual Representation of the Generic Coordination Model

from the receiving department (n_k); *or* when an invoice arrives from the vendor (n_m).

Graph models provide a formal basis for defining control flow; they also provide a set of natural graphic representations. It can be shown that for graphs such as Petri nets, there are *many* different graphic visualizations [37]. For example, Figure 1(a) is a conventional timed Petri net representation of the simple precedence relation, (n_i, n_j); Figure 1(c) describes the more complex one where $(\{n_j, n_k\}, n_i)$ and (n_m, n_i) are both part of the precedence specification. Figures 1(b) and 1(d) are an alternative representations of the same situations. Notice the way that sequential, OR, and AND control flow logic are represented using the visual control flow graph Figures 1(c) and 1(d).

The specification represented by Figure 1 provides a static description of the precedence. In Petri nets, dynamic execution is represented by using *tokens* to represent how workcases flow through the steps; when a workcase is being processed by a step, the token corresponding to that workcase is placed on the node corresponding to the given step. Tokens flow from node-to-node with no other explicit coordination if they are connected by simple edges. (This is the attraction of models like Petri nets for expressing coordination strategies.)

Tokens that traverse a multi-headed arc are replicated and sent to each node at the head of an arc; a token cannot be delivered to the single head of a multi-tailed arc until a token is released from each tail node. The token/workcase distribution on a static coordination graph is a snapshot of the execution of the static graph. By looking at a sequence of snapshots, e.g., creating a new snapshot each time a token moves from

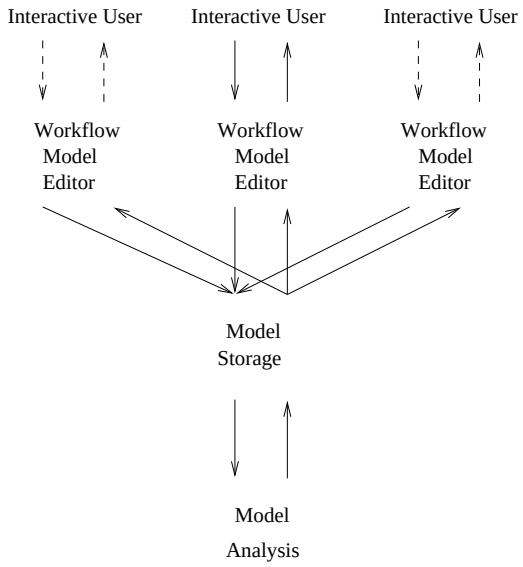

Figure 2. Basic Workflow System Organization

one node to another, it is easy to animate a display of the static graph to illustrate the dynamic behavior of workflow in the procedure [38, 39].

It should be noted that in pure Petri nets, the choice of paths through which tokens flow in an "execution" of the net is nondeterministic. In variants such as WooRKS [40], control flow path selection is made deterministic through the addition of supplementary information. Zisman's Petri net model [17] uses rules to augment the normal Petri net coordination specification, though not explicitly to achieve deterministic control flow as is the case with E-nets [41] and other timed/colored Petri nets [42].

2.2. Workflow Modeling System

The use of workflow languages for representation, analysis, and design has led naturally to the need for computer support. A *modeling system* provides facilities for creating and browsing a representation model, for applying various algorithms to an analysis model, and for collaborative interaction and information archiving for design models (see Figure 2) [28, 29, 38, 43, 44].

The modeling system is first an editing environment in which a workflow specification can be prepared. Visual workflow models are most often supported by providing a point-and-select graph editor. The editing environment must provide an appropriate mechanism for allowing the user to define and browse hierarchical models, where the refinement is a linear program or a nested graph model.

The modeling system provides a means for saving a model once it has been defined. The internal format of the model is used by any other tools that are incorporated into

the environment, e.g., analysis tools.

Finally, it is important that the modeling system support multiple users at the same time. This allows a group of users to collaboratively design and/or study a model of a procedure.

Workflow modeling systems are an invaluable tool for studying procedures. They can be the basis of business process reengineering studies, since they provide an explicit model for representing and designing business procedures.

2.3. Some Example Modeling Languages and Systems

People have built many different modeling languages and systems using Petri net coordination language with various computation languages.

E-Nets. Our own earliest example of a language combining a computation and a coordination model was the E-net modeling language [41]. It used timed Petri nets with a special-purpose, interpreted computation language explicitly intended to represent systems. An E-net could be analyzed for various performance characteristics. The application domain for E-nets was computer simulation rather than office procedures, but the underlying technology was the same as for workflow languages. No system was ever built to support E-nets.

ICNs. The information-control net (ICN) language was explicitly designed as a workflow language to represent and analyze office procedures [45, 18]. The ICN had a control flow graph (isomorphic to the E-net coordination language), coupled with an information flow graph. More precisely, an ICN node set is the union of a set of step nodes, A, (called activity nodes in the language), and a set of data repository nodes, D. Edges among members of A are precedence edges, as explained above. Let aϵA and dϵD; then an edge (a, d) represents the case that when the computation associated with step a executes, it may write information to the data repository d. Similarly, if (d, a) belongs to the edge set of the graph, then when step a executes, it may read information from data repository d.

As we derived ICNs from E-nets, we were concerned about how natural these models might be for uninitiated office workers to use. Ellis and Morris conducted an extensive field study to determine acceptable visualizations to represent the coordination formalisms, resulting in the specific visual appearance of the ICN model [46]. In these field studies, it also quickly became clear that even though colored Petri nets were sufficient to represent every situation we could think of, office workers felt much more comfortable with subsidiary data flow graph described in the previous paragraph.

The Quinault modeling system was designed to support the representation and

analysis of ICNs [39]. Quinault was a point-and-select, visual systems enabling its user to create, edit, and animate ICNs. The follow-on system extended Quinault's functionality across a network of workstations, establishing the basis for collaborative model design [38]. Olympus is a more recent distributed analysis and design system supporting ICNs [47, 48].

ICNs were ultimately used as the base language for the Group Bull FlowPATH workflow enactment product [49], described further in Section 3.

IBM FlowMark, FileNet Visual WorkFlo, and Action Technologies Action Workflow are full workflow enactment products that have a modeling language and system component.

IBM FlowMark. Models in FlowMark are a combined data and control flow graph [21, 28]. The control flow graph specifies activities, processes, connectors, transition conditions; the data flow graph represents input and output containers with data mappings to activities. There is a formal underlying language, the FlowMark Description Language, for each visual model constructed in the system. The model can be used for representation, analysis, and design.

FileNet Visual WorkFlo. This commercial product has been around for several years, and is especially well-known for its early distributed technology [44, 50]. The Visual WorkFlo/Composer is a graphical package used to define and analyze business procedures using a precedence flow graph. Part of the information flow is specified by the graph, and the remainder is provided by an "instruction sheet" to define routes (among a set of alternatives in the graph) along with processing actions that need to occur as the work flows through the procedure.

Action Workflow. The ActionWorkflow product [43, 51] has earned a reputation for supporting workflow technology based on its underlying "speech-act" model of how people work with one another [52]. The system is designed on the premise that when work is to be conducted, there is someone who wants the work done, and someone else who will perform the work [43]. The Workflow Analyst is the component of the system that is used to model procedures. The tool enables end users to construct a speech-act model of the work that can be used to describe the work to all interested parties, and to perform basic analyses.

The Process Handbook and PIF. The PIF effort [6] provides a metalanguage in which one can represent different coordination and computation models. Malone, et al., use PIF as the basis for the Process Handbook to describe different procedures that might be reused in different environments [7].

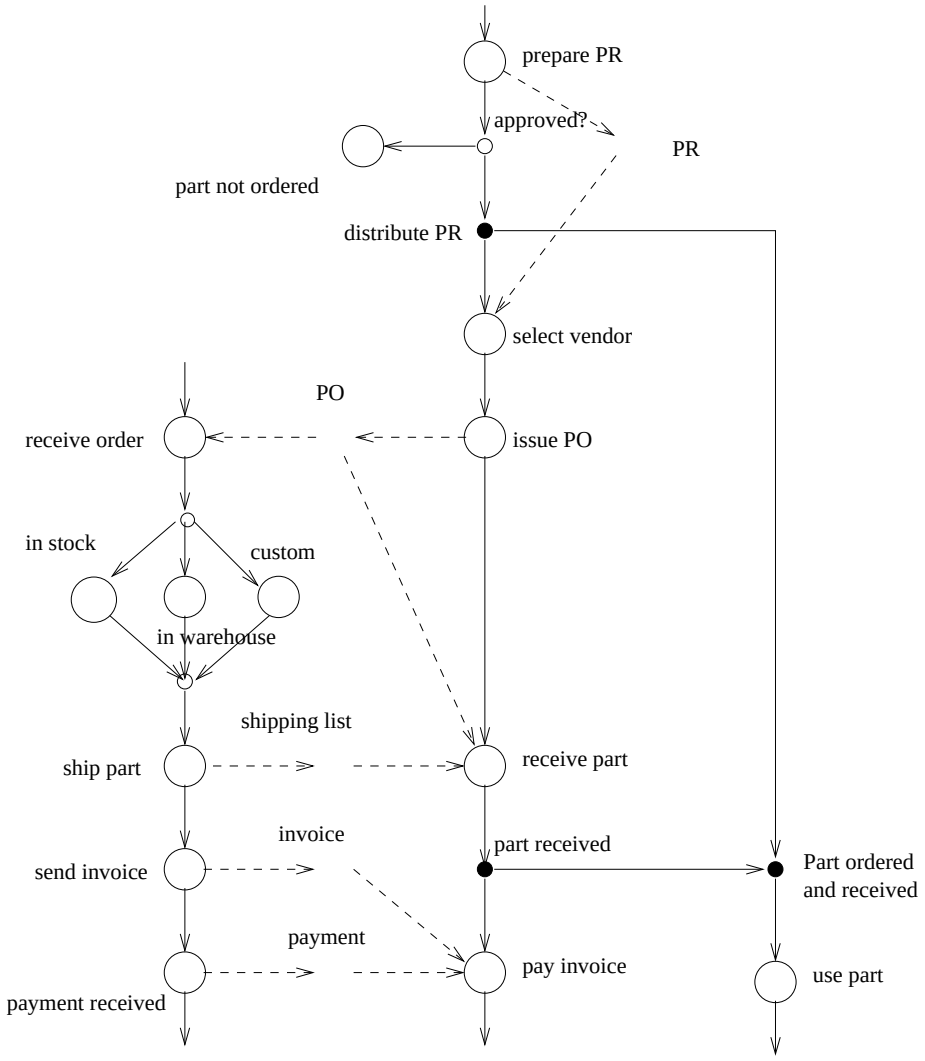

Figure 3. An ICN of the Purchasing Procedure

2.4. A Model of the Purchasing Procedure

Figure 3 is an ICN model of the purchasing procedure introduced in Section 1; models in the commercial systems mentioned above are semantically similar to the model described here. In ICNs, circles represent steps in the procedure, and solid edges represent precedence among the tasks. There are three kinds of steps (called activity nodes) in an ICN: a large circle represents an ordinary processing step; a small, open circle represents disjunctive (OR) control flow — a place where a decision is made regarding control flow; a small, filled circle represents conjunctive (AND) control flow — a place where concurrent activity can be initiated/synchronized.

In the figure, the "approved?" OR-step occurs after the "prepare PR" step has

terminated; when the "approved?" step finishes, *either* the "part not ordered" step or the "distribute PR" step is executed next, depending on the outcome of the approval. The "distribute PR" step following the "approved?" step causes the "select vendor" step to be enabled and the "part ordered and received" step to become only partially enabled (since it also depends on the completion of the "part received" step.

The rectangular nodes are data repositories, and the dashed edges between steps and repositories represent data flow into and out of a step from/to a repository. The "prepare PR" step writes into the "PR" repository, and the "select vendor" step reads from the PR. Similarly, the "PO" repository is read by the "receive order" step (in the vendor's part of the workflow model) and by the "receive part" step; the "issue PO" step writes this repository.

2.5. Issues in Workflow Models

There are some outstanding issues in the way workflow models should be handled by a system. Here we summarize those issues.

Language Constructs. Workflow models have generally converged on the notion of a coordination language and a computation language. Independently, different studies have moved toward control flow with sequence, conjunction, and disjunction. The multi-edge coordination model allows one to represent sequential constraints, alternative paths, and logically concurrent operation among a set of tasks. Coordination models typically conform to Petri net control flow semantics. While the Petri net control flow semantics are sufficient, they can sometimes be difficult to use to represent a specific situation, e.g., where one would like to fire a transition if a condition is false rather than true. (In the Petri net research community, this problem was addressed by adding *inhibitor arcs* in some variants of the language [36].) Some users continue to feel unduly constrained by the language syntax, though their concerns can sometimes be addressed by revising the syntax of the underlying model [37].

Handling Detail. Workflow modeling languages are frequently visual to enhance their utility, causing scaling to be a major concern. Scaling can be addressed by using hierarchical descriptions. The modeler identifies steps in the procedure, then provides an interpretation of the step through a conventional linear computation language, or through a nested single-entry single-exit graph model (e.g., see [53, 54]). If flow detail is important in the representation of the model, it is reflected as a set of steps within the coordination model; if it is desirable to ignore detail within the context, detail is abstracted into a single step in the model.

Within computational model step specifications, abstraction can again be used to

suppress detail. This is accomplished using traditional software modeling techniques: the interpretation can either hide details through abstract machine hierarchies (functions) or by simply ignoring some characteristics. Hence, this class of models provides a spectrum for incorporating or suppressing detail in the procedure model specification.

Submodels. Workflow systems need to provide ways for an analyst to reuse submodels. Often a user defines a submodel, then wants to reuse it in different parts of the overall model. For example, a submodel may describe the steps to obtain supervisor approval for some action; this same set of steps may be used at various points in a procedure. In some cases, conventional macro semantics are exactly what is desired since there is no intent that there be interaction among the different copies of the submodel. However, in other cases, is is desired to explicitly centralize the function described by the submodel into a single entity — one representing a single object used by many different parts of the model. Most languages do not allow the model designer to reuse submodels by instantiating copies; support for a single, shared submodel may also be ignored.

General Issues. Research is workflow modeling languages is less intense now than it was in 1980. Within the class of workflow approaches, there is reasonable convergence on the idea that the language needs to have a computation and a coordination component. Independent of the issues relating to scaling visual models, selecting the most appropriate syntax for a domain, and reusing submodels, most of the current effort in this area is to address issues relating to how the modeling environment should fit into the enactment environment.

3. Workflow Enactment

Workflow *enactment* refers to the situation where a procedure is encoded into a set of directives that will be executed by some combination of humans and computers in a workflow management systems [5]. Workflow is intended to be used in a traditional topdown design methodology: a model is derived to analyze the procedure; the coordination and computation specifications are refined during the design phase, deriving the coordination specification for the implementation; during the implementation phase, the step interpretations are fleshed out by refining the computation specifications for each step to produce a specification suitable for enactment.

3.1. Refining Workflow for Enactment

The coordination specification for a detailed design will have identified the steps in the target procedure, and have established the order in which steps should be executed for each workcase. At design time, the computation specification for each step is intended to represent relationships, constraints, assertions, etc. At the time the step is implemented, the step specification must be replaced by a software module that implements the step.

In operating system terms, a step is a schedulable unit of computation. When a workcase needs to be operated on by the step, the unit of work is assigned to a processing unit capable of executing the computation specification — usually a human using a computer, but sometimes to an unattended server machine. The workflow enactment system provides a multitasking environment to manage the associated resources and to coordinate (schedule and synchronize) step execution.

3.1.1. Step Computation Step computations are general-purpose computations, implemented either as unattended or interactive processes.† As a result, the computation language needs to incorporate facilities for general-purpose computation (e.g., arithmetic and control flow) [26]. All workflow processes execute on network elements, so the step computation specification needs to use network protocols to access information within the workflow environment. Interactive processes must incorporate software to implement a human-computer interface, allowing a user to control step execution.

The result of these requirements is that the effort to implement an enactment system is orders of magnitude more difficult than that of implementing a modeling system.

3.1.2. Step Coordination A basic workflow modeling specification represents a procedure independent of the way the components of the work will be assigned to processors. Resource utilization is not explicitly taken into account; rather, if the analyst thinks that resources are critical to the way the work is to be performed, then the model must describe the details of resource use. For example, this can be accomplished by having the program prescribe when a specific processor is to execute a step. A consequence of this approach is that tokens in the workflow graph may represent both workcases and processing agents (called *actors* in the workflow literature). While this modeling approach may be representationally complete, it has three significant negative aspects: (1) it complicates the expression of the office procedure, (2) it makes it difficult to assign a single actor to multiple roles, and (3) it does not provide

† Any specific workflow system may be implemented in terms of objects, threads, processes, or other schedulable unit of computation. For simplicity, we will refer to these schedulable units of computation as "processes" in the remainder of this paper.

any obvious mechanism to represent actor scheduling/preemption (to multiplex across pending work). Therefore, modern workflow systems use the analogue of a Petri net token to represent a workcase, then represent actors using other model atoms, e.g., see [21, 28, 55, 56].

Actors may be people or computers; their *roles* identify a set of activities that the actor is capable of performing (i.e, an actor represents one "processing entity," so it can only do one thing at a time) [6, 8, 11]. In modern workflow enactment languages, tokens represent only transactions (workcases) while resources, including processing agents, are represented using role and actor constructs.

The role identifies a particular type of processing, e.g., a purchasing agent, but it does not identify the specific person or computer that can/will execute the role. Assignment of a processing entity to a workcase and step is done by explicitly *scheduling* an actor to a role for a workcase. Thus the enactment specification associates (map) steps and roles, identifying the schedulable unit of computation. Then a runtime component associates actors with roles, meaning that any such actor is capable of playing the role to which it is associated. When a step is to be applied to a workcase, the runtime system selects an actor capable of playing the role containing the step, then assign the step-workcase to the actor. Of course, any given human actor can hold many different roles at a time (time multiplexing across them), such as purchasing agent, supervisor, employee, etc.; so the actor-role mapping is a many-to-many mapping.

When the workflow system is put into operation (for a particular procedure), there can be many different workcases in progress at any given time. Further, since workflow coordination model support conjunctive control flow, different parts of a workcase may be executing concurrently. This, of course, requires that the system synchronize the operation according to a concurrency specification. In a workflow system, the coordination model explicitly defines the synchronization strategy. Note that the details of the strategy depend on the semantics of the coordination model; e.g., colored Petri net semantics ensure that as different workcases flow through a set of steps, they do not interact with one another, (synchronization operations only apply to conjunctive control flow within a single workcase) [42].

3.2. Workflow System Organization

Figure 4 represents a typical workflow system organization. The coordination model interpreter is the centralized component of the system that decides how work will be scheduled for the rest of the system on the basis of the coordination model specification, the actor resources available, and the work load (workcases).

When a workcase arrives, the coordination model interpreter determines which step will first process the workcase, determines the role to which the step belongs, then

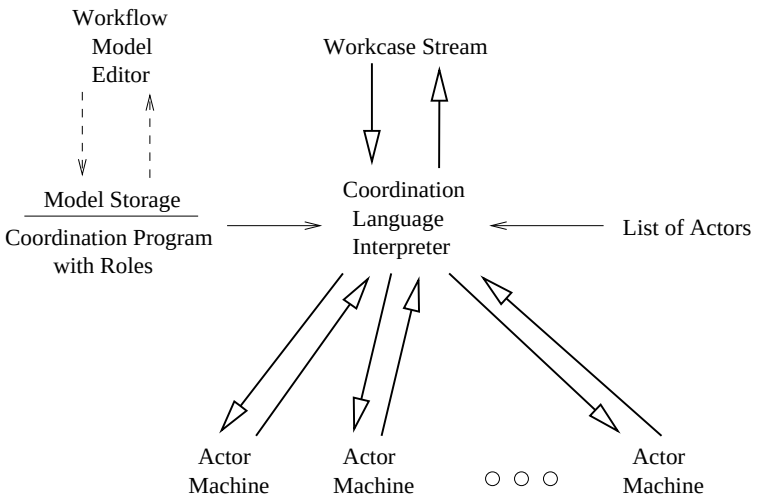

Figure 4. Workflow System Architecture

assigns the work to a corresponding actor. Work assignment is accomplished by sending a workcase to the actor's machine. Workcases can accumulate in an "in-box" for the actor. The actor eventually selects a workcase, then applies the step computation to it (in the context of the actor's computing environment). Since the work is ordinarily intended to be performed by the computer and the human, the computer prompts the user for input whenever it is needed. The human user determines the appropriate response — perhaps using other tools in the local computing environment — then responds to the computer (local execution of the step computation). When the step has completed its computation on the workcase, the result is reported back to the coordination model interpreter for subsequent scheduling.

3.3. Some Example Enactment Systems

There are many examples of workflow enactment systems, including Prominand [57], Domino [58], the C&Co enhancement to C [26], Visual WorkFlo [44]. The Workflow Management Group is a manufacturer consortium organized to negotiate common ground on the various commercial offerings [11] (the PIF [6] and Process Handbook [59] are also concerned with uniform representations). Also see [5, 9, 20, 21] for surveys of commercial products.

FlowPATH. As mentioned earlier, the ICN model was used as the basis of the Group Bull FlowPATH workflow product [49]. FlowPATH uses a logically centralized server with a relational database to store the model and the data for the procedures. Step interpretations are executed as processes in client workstations. In our experience with

the FlowPATH development, we learned that the vast majority of the engineering effort was in constructing facilities so that information stored in the server was accessible from various client workstations on various servers, and so that the system had an acceptable human-computer interface. The workflow approach was the crucial vision for the system design, but the majority of the engineering effort was invested in technologies needed to support any interactive, distributed software.

FlowMark. We previously introduced IBM FlowMark in Section 2 [28, 21]. As an enactment system, FlowMark is a distributed system with a server to store the model and common data, and clients to manage the "buildtime" (modeling) tools, the "runtime" tools, and the "administration" tools. The runtime part of the system coordinates step execution by staging work and monitoring progress. The administration tools are use to setup the system and to manage an audit trail of work in the system. Like FlowPATH, there is a significant set of underlying software available to step execution software via a packaged application programming interface. Actors and roles are also used as part of the runtime system's scheduling information.

InConcert. The Xerox XSoft InConcert workflow product provides similar capabilities to FlowPATH [56, 29]. It is based on a workflow graphic model to define steps and their coordination model, including the flow of information and the role of people in the procedure, to define an enactment system. In InConcert a graph editor is used to describe how a procedure should be implemented by defining jobs, tasks, roles, and the flow of jobs among tasks for different roles. Based on the graph definition, the system generates appropriate software to route work and to reference data stored in an underlying relational database. InConcert also allows its users to integrate their preferred productivity tools, and to override the default flow pattern.

ActionWorkflow. ActionWorkflow supports the design of the speech-act model of work (see Section 2) as a composite model of standard interactions [43]. Once the model is designed, the management system supports the enactment of the model by moving work requests to work suppliers, by providing the proper tools and information to accomplish the work, then by routing the work to the next supplier. The management system also provides various support tools to remind work suppliers of pending work, etc.

3.4. Workflow Enactment of the Purchasing Procedure

In Section 2 we provided a descriptive model of the purchasing procedure. While a full enactment specification is beyond the scope of this paper, we can briefly consider a specific part of the purchasing procedure to get the flavor of an enactment system and

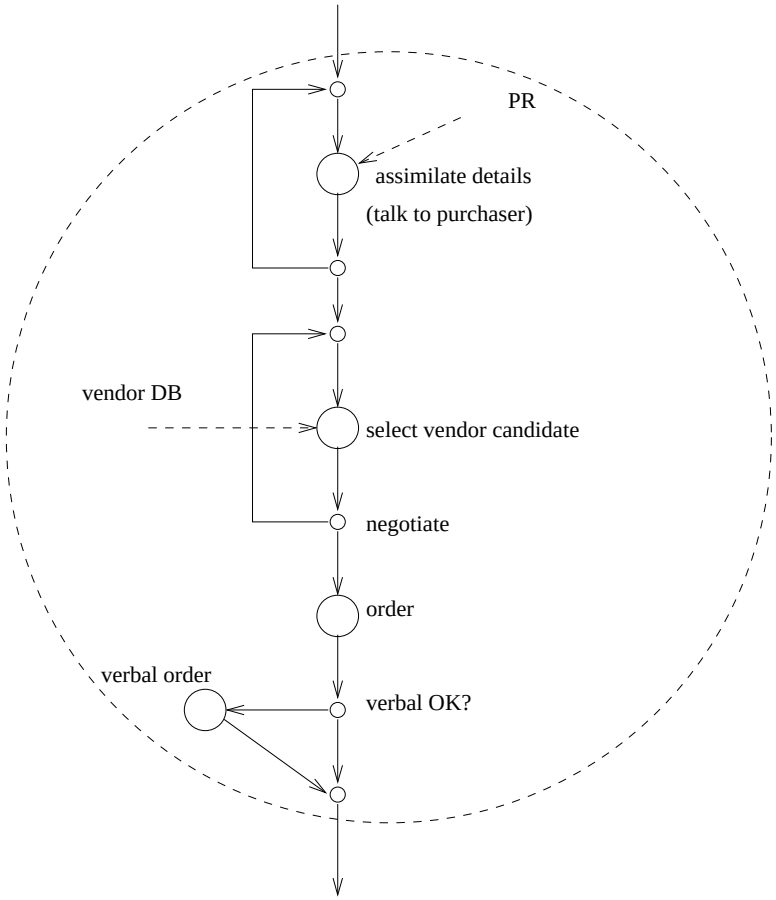

Figure 5. A Nested ICN to 'Select a Vendor'

specification (using the ICN model and system).

The ICN-Olympus computation language uses nested ICN graphs and C programming language functions to specify step interpretations. The coordination language is the same one described in Section 2, complemented with actor and role specifications. Figure 5 represents a refinement of the step to "select a vendor" as another ICN with single control flow entry and exit edges. As expected, this refined graph appears much like a flowchart for a serial program (though we do not require that the nested ICN be serial).

A step node is activated by the presence of one or more tokens flowing into the node. In the enactment system, each token has attributes of the workcase that it represents, the workflow system must pass the attributes to the textual interpretation. Each ICN node interpretation receives an input parameter of type `USER_INPUT` holding an encoded form of the token attributes; every interpretation (in the Olympus implementation) is of the form shown in Figure 6.

The `task_return` is a call on the workflow runtime system to pass the output token

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

Figure 6. A Step Interpretation in C

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.

The runtime environment provides a variety of facilities to support step interpretations, including graph manipulation and interpretation, workflow entity management, workcase manipulation, and scheduling (see [22] for more details for ICN-Olympus, or vendor materials for other systems).

3.5. Issues in Workflow Enactment

The Nature of the Specification. A key aspect of workflow enactment systems relates to the nature of a particular workflow specification: if all steps can be completely automated, the workflow specification is simply a style of high-level distributed programming. However, in those cases where humans must be involved in the processing of individual steps in the procedure, the workflow program explicitly coordinates a human's activity with the activity of the machine and other humans. That is, it defines a means to coordinate an individual's work to accomplish collaborative work. Thus a problem in workflow enactment languages and systems that distinguishes it from the mainstream of distributed programming languages and systems is that it provides a distributed computing environment in which humans collaborate (using the system) to execute a procedure. In particular, workflow is based on a coordination specification for step execution, which essentially schedules human activity to make it conform to the procedure's coordination model.

Work Specification. A related problem is that when work is assigned to a user at a workstation, the details of the assignment must either be implicit to the procedure, or explicitly encoded into the step computation implementation. For example, if the

execution of step W is assigned to user X at machine Y for workcase Z, how will the user know what to do?

The Social Model. There is overwhelming evidence that a pure workflow approach dramatically influences the social model among an organization's workers, (see Parts I and II of [60]). The impact of the technology on the behavior of the workers can be so severe as to cause workflow systems to completely fail (e.g., see [2]). Situated work advocates argue that the coordination component of workflow — the aspect that distinguishes it from systems supporting situated work — is incompatible with social models used in the workplace.

Manifestations of the Issues. The manifestations of these problems are diverse; workflow is often criticized as being:

Too Prescriptive. Often a workflow program over-constrains the solution by providing coordination or computation constraints to the human worker that prevent him or her from simply performing the work in the manner most familiar or comfortable.

Too Vague. The work specification does not provide enough information to the workers to enable them to perform the desired work.

Exceptions. If a workcase does not happen to conform to the expected format for incoming workcases, the procedure may be unable to adequately handle the workcase. For the human worker, this can mean that the computer schedules work for them, placing the human in a state where it is impossible to execute a step. Exceptions occur because the procedure is an incomplete specification of the intended processing, because the intended procedure itself is incomplete, or because the procedure is completely inappropriate for the work [61]

Change. A workflow procedure is specified, but the procedure must change frequently due to changes in the nature of workcases or the computing environment.

Workflow as fiction. The workflow specification may be the correct way of performing work "by the book," but it may not represent the way people actually perform the work. Forcing humans to do the work according to an arbitrary operational model is too rigid and inefficient. (This is a variant of the "too prescriptive" problem.)

Loss of informal information. When a manual procedure is encoded as a workflow procedure, the encoding will almost certainly fail to encode *all* the information flow associated with manual processing. The manual information flow reflects the social

relationships as well as the formal organization of the group; information flow in the social model is easily lost.

Loss of informal processing. The logical extension of ineffectiveness due to informal information loss is ineffectiveness due to the loss of informal work. Social organizations recognize where there are flaws in a procedure and the team members compensate for failures in the procedure through informal activity. By formalizing the coordination and step definition, there are fewer opportunities for human workers to compensate for inadequacies in the process.

State-of-the-art workflow enactment systems [5, 21] do not address the issues described here [1, 2, 12, 19, 62]. One solution to this problem is to remove the coordination component from the system; this amounts to taking the situated work approach. However, all organizations impose at least some minimal set of procedural guidelines on how its workers perform their work. For example, a corporation has specific guidelines for authorizing and paying for travel; an insurance company has a specific procedure for handling a claim; and government procurement officers have a strict set of guidelines defining how they solicit bids, how they select vendors, and how they negotiate a contract.

The challenge to modern collaboration supports systems is to design languages and systems that preserves group work styles (social models) built through informal communication and understanding, and which will support styles ranging from cases where situated work is the best approach to those where strict coordination is required.

In the remaining sections of this paper we discuss how workflow technology will continue to evolve to meet the challenges encountered by state-of-the-art systems.

4. Evolving the Workflow Enactment Language

Workflow specifications cover a large space of approaches, ranging from the informal to the formal, intuitive to detailed, declarative to operational, and descriptive to prescriptive. In Section 2, we argued that models tend to vary in detail; an enactment specification to be executed by a computer is necessarily operational and detailed since it defines all processing activity related to the step. What form should the part of the specification take that directs human activity? A declaration of the intent of the step might be more appropriate than an operational spec of how the step can be accomplished.

Historically, workflow developers have adopted a single approach whereby the step specification has a specific level of detail in the description, usually expressed as an operational description rather than a declarational one. Conversely, planning systems sometimes make heavy use of declarational specifications that can be analyzed to infer

the step computation, and, depending on the extent of the specification, even infer the coordination of the steps (e.g., see Polymer [63]).

4.1. Generalizing the Enactment Language

It is clear that there must be workflow languages that can address a variety of approaches along different dimensions. In particular, the language should be able to express high and low levels of detail for the step specification; it should be possible for the specification to be provided as an advisory note, or as a required way to perform the work; the nature of the specification should range from operational to declarational. In this section we explain why these characterizations are important to workflow languages, then we define a 3-space of languages. The ideal workflow language would allow a specification to have different steps be defined in different parts of the space, depending on the way the steps are intended to be interpreted.

4.1.1. Conformance In a workflow environment, an end user's perspective of the work is that workcases are routed to that user, the steps in the role are to be applied by the user (with the aid of local computing facilities), then the workcase is to be returned to the workcase scheduler so it can be assigned to another actor.

The user has two basic types of information to assist in executing the steps in the role:

- Workcase identification and step computation descriptions. The computation will usually have been encoded so that a process executes in the user's computing environment, interacting with the user via a human-computer interface. For example, in FlowPATH, a workcase and step identification is placed in the user's in-box. When the user decides to execute the step on the workcase, he or she runs the step computation; this program creates a procedure-specific human-computer interface where the process executing the step and the user interact to accomplish the work specified by the step computation specification (the "step program").
- A description of the coordination component of the workflow model. This allows the user to infer how the workcase has been processed before it arrived at this station, and where it will go after the user has completed processing it. For example in FlowPATH and InConcert, the user can view the coordination model at any time.

Any other information required to successfully execute the step has to be a part of the environment in which the user performs situated work (e.g., it might be a conventional electronic desktop with a spreadsheet, a communications port to a mainframe computer, etc.).

How well does this work? Using a conventional workflow approach, we will encounter several manifestations of problems associated with staging work for humans. If the step

program is too prescriptive, then it is difficult for the user to perform the work; the user will learn to bypass the system. If the step program is too vague, the system provides no service other than reminding the user when he or she has work to do.

The workflow system designer may design the system so that it requires the workcase be executed according to the specification. Such a system will have difficulty coping with exceptions, will encourage loss of informal information and processing, and will generally work against the elements of the group's social model. We say such systems force high conformance to the workflow specification. High conformance is desirable in cases where the organization has hard constraints on the way the procedure is to be accomplished.

At the other end the spectrum are systems where the step program is advisory, meaning that it indicates what should be done, but it does not really assist in performing the work, nor require that the user actually perform the work in a way that it complies with the model assumptions. We say that such a system does not require that the user conform to the specification at all. The advisory procedure definition amounts to an example of how the work might be done to achieve the goal the designer has in mind.

4.1.2. Level of Detail Interactive step programs accomplish the step's work through a combination of effort by the user and computation by the computer. There is another spectrum describing how much of the detail of the step computation is encoded into the computer program, and how much is left to the discretion of the user.

Programs with a moderate amount of detail provide little assistance to the user; they are often described as being too vague to be of any real use. There is little chance that they will incorporate any significant amount of informal information or processing. We say they have a low level of detail. A low level of detail specification is sometimes a necessity if the spec writer cannot predict all cases that might arise at execution time.

On the other end of the spectrum are coordination and step programs that are highly detailed. The detail tends to cause the system to constrain the user too much, resulting in the system being perceived as being too prescriptive, unable to cope with exceptions and change, and discouraging informal work. This style of program has a high level of detail. In cases where the actor is a novice, highy-detailed specs are desirable.

4.1.3. Operational and Declarative Specifications The language of step programs could focus on *what* is to be accomplished by the resulting program, not on *how* it is implemented. Such requirements are said to be *declarative* specifications identifying what the procedure should accomplish. However, enactment design specifications are often *operational* ones that define how work is to be accomplished (as opposed to what is to be accomplished). Computer programs are usually operational specifications, e.g., they choose an algorithm to accomplish some specific task. Parts of the step program that define the human-computer interaction will generally be more useful if they are

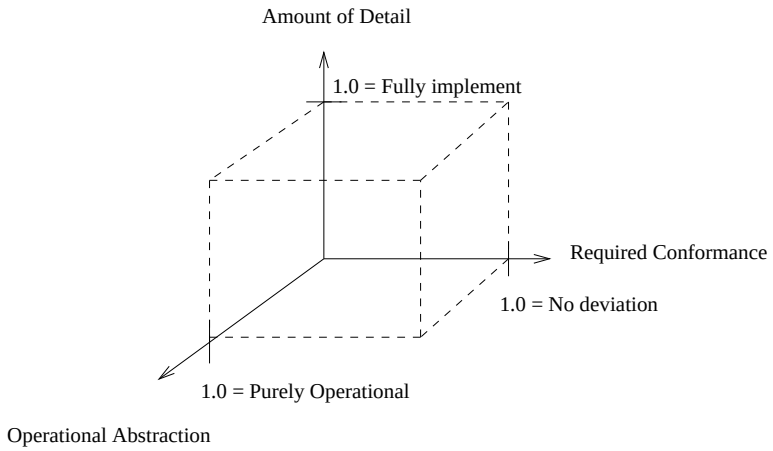

Figure 7. The Model Domain Space

an expression of the intent of a step rather than being an operational specification. However, operational specs are easier to write, and can be used with a low-conformance approach to achieve a similar effect.

The tendency to use operational specs throughout the procedure specification is a fundamental problem with many workflow systems; the step programs are generally operational, when they should sometimes be operational, sometimes declarational, and sometimes just be a simple statement of intent. Such an approach is insufficient for generic software design, but workflow programs should *not* be treated as traditional software systems; part of the ultimate implementation of the procedure is human work and its details should be left to the human participant for each individual workcase.

4.1.4. The Model Space A workflow model can represent parts of the process that *must* be performed for the process to be acceptable (e.g., the part of the work that is controlled by federal regulations), and parts that *may* be performed by a computer or human (e.g., a mechanism to describe a recommended way of accomplishing the work). The language might also allow one to represent parts of the process at very abstract levels or very detailed levels. Along a third dimension, a specification may support representations that are highly operational or highly declarative; goal and intentional specifications are considered to be highly declarational and a C encoding of a sorting algorithm is considered to be highly operational.

Thus, models should represent procedures according to the way the model is to be used, as defined by three different criteria: the amount of *conformance* that is required by the organization for which the process is a model, the *level of detail* of the description, the *operational* nature of the model (i.e., to what degree does the model describe how the process works rather than what is required from it (see Figure 7)). In

this domain, models that represent only *structured* or *explicit* work (e.g., [49, 64]) are in a subspace where $x \rightarrow 1$, $y \rightarrow 1$, and $z \rightarrow 1$; systems intended to address *unstructured* work are in a space where $x \rightarrow 0$, $y \rightarrow 0$, and $z \rightarrow 0$. The domain for which descriptive and analytic workflow models (e.g, [45, 65]) was intended is the plane defined by $0 \leq x \leq 1$, $0 \leq y \leq 1$, and $z=1$. Conventional workflow enactment systems [8, 49, 21, 56, 44] could be characterized as a line segment in the space with $x=1$, $y \rightarrow 1$, and $z = 1$. Systems that focus on exception handling [61] are in a space where $x \leq 1\text{-}k$, $(0<k<1)$ and $0 \leq y \leq 1$ and $0 \leq z \leq 1$. Goal-based systems (e.g. [32, 66, 63]) are in a domain in which $z=0$, but x and y vary according to the specifics of the model; e.g., an inferencing system assumes that $y<1$.

4.2. Extending the ICN Model

Based on the observation that people sometimes prefer to work through *goals* rather than through procedures, we introduced *goal nodes* into our ICN model [67]; a goal node represents a part of the procedure with an unstructured work specification. The details of the step are described only in terms of goals, intent, or guidelines; however, the task appears in a framework of an operational coordination model, so that the conditions under which it is executed are tightly specified. The thrust of goal nodes is to enable one to use operational specifications for coordination, but to be able to use informal specifications to define individual task computations, particularly manual tasks. This approach allows a workflow designer to provide structure where it is needed, but to allow for computational specifications that are more acceptable to human workers when the actor that executes the task is a human.

The goal-based ICN model addresses the full space, with different parts of the model addressing different subspaces according to the need for that part of the model. For example, if part of the work is highly structured operationally specified, and required to be accomplished according to the specification, (as is the step to "issue a PO" in Figure 3), then it should be modeled differently than work for which only the goal is known (as is the case in step "select vendor" in the same figure). The model should allow one to represent a process for which parts are operational and required, while the way that other parts are executed is arbitrary, provided that the executions satisfy the intent.

In an extended ICN, each step belongs to a *region* whose type is defined by a point in the space in Figure 7. For example, a type "R" (for required work such as "issue PO") region is represented by a point in the space at (1, 1, 1); the model can be represented as a conventional ICN subgraph composed entirely of operationally-specified steps.

A type "A" region (for assisted work) also uses an operational-style specification, but the submodel in the region can be interpreted as one approach to accomplishing the work (in the absence of a declarational specification). The "prepare PR" step in

Figure 3 is a good candidate for this class of steps. This type of specification is a point in the space such as (0, 1, 1); it is used when a process designer has one notion of how to accomplish the work, but is not willing/able to abstract the steps into a declarative form; the A-region work can be used directly, or it can be used to (manually) infer the intent of this part of the work. This is how traditional (R-region) workflow models are often used — as an example of how work might be done.

A type "D" region (for declaration region) represents a part of the model that defines *what* the region is intended to accomplish, rather than a description of *how* the work must/might be conducted. The step "pay invoice" is a good candidate for this type of region (occurring at (0, 1, 0)).

A step in a type "G" region (for goal-based region) is one for which no interpretation is assumed, e.g., the "approved?" step in the example; these steps should be interpreted with the semantics of the point (0, 0.25, 0), meaning that the step specification lacks detail compared to a D-region, but specifies the goal and intent of the work in the region.

The rationale for extending the model over the conventional workflow approach is to be able to address different points in the space of solutions shown in Figure 7. The empirical evidence reported in [60] and in [2] as well as in countless CSCW papers argues that CSCW systems should support various points in the space. We find most of these arguments to be creditable, though each is driven by a particular scenario or set of circumstances. For computer systems to provide value to a group, it must be designed so that all points in the space can be addressed. A language that ignores large parts of the space, especially an entire dimension, are likely to be criticized on the issues raised in Section 3.

The cost of extending the language to accommodate these various perspectives is modest — goal nodes and regions specifications. The main incremental cost is in the workflow system design. In the next section, we will describe how a system can provide support for goal nodes.

5. Evolving the Workflow System

A flexible language is a necessary but not sufficient part of an effective workflow system. Given a flexible language such as the extended ICNs, workflow enactment systems must also be broadened from their current form to take advantage of these language extensions. They must also provide increased flexibility in the style of collaborative work they supports.

As described in Section 3, a conventional workflow enactment system is composed of a set of interpreters for the coordination and computation models. The coordination model interpreter accepts workcases, then schedules their flow among various machines that execute steps. The presence of actors and roles adds a level of scheduling to the

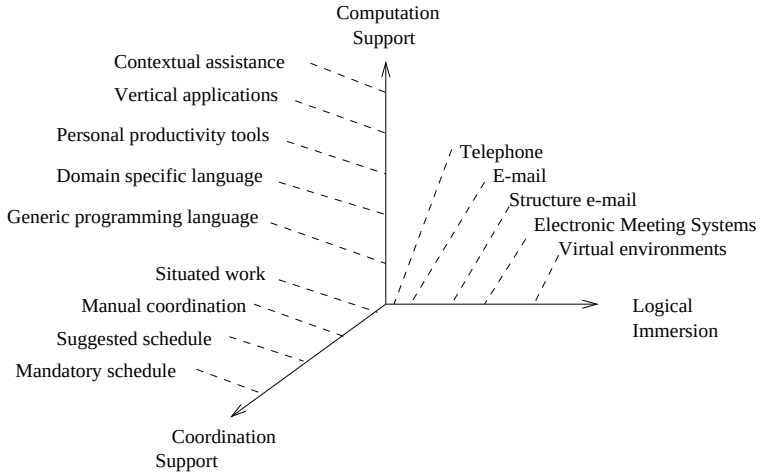

Figure 8. CSCW Characterizations

problem, since once a workcase has been assigned to one actor within a role it usually (but not always) must be scheduled consistently with the same actor as long as it is within the scope of the role. After a step has been assigned to a computation machine and has been executed by an interpreter, the result is reported to the coordination model interpret so it can continue to schedule the workcase for subsequent steps.

The enactment system must use the enactment language to further address the issues identified in Section 3. Extended ICNs identify regions of a workflow spec requiring different kinds of support; the system must follow through to support assisted work, required work, and provide the appropriate mechanisms for managing goal node execution. The workflow system must also explicitly address the general characteristics of a collaborative system, otherwise it will ignore the general issues we have raised. By understanding those issues, it is possible to design a broader class of systems that meet the needs of the team of workers.

Figure 8 is an informal characterization of different CSCW system approaches. The x-axis informally represents the amount of *logical immersion* a human worker has in the system. The y-axis describes the amount of *computation support* the system provides human users. The z-axis represents the amount of *coordination support* the system provides. Generalized workflow enactment systems should take all these factors into account, as we explain in this section.

5.1. Logical Immersion into the Environment

The x-axis describes the amount of logical immersion each human has in a collaboration session. Near the origin is support of the collaboration using the telephone; the environment among the collaborators is loose and informal, with communication

occurring via the telephone.

Electronic mail systems logically immerse the collaborators deeper into the system. Whereas telephones are usually a distinct device from the computer and network, electronic mail is a part of the computer facilities, e.g., always available to the worker in an alternate window. Some systems add a layer to electronic mail so it can be used to capture some of the context of a domain. For example, the mail system delivers work to a person, and routes work onto others once one person has finished with it (see Chapter 8 in [60] as well as various commercial products).

Electronic meeting rooms represent a higher level of immersion into the computer system. One class of electronic meeting rooms is an extension of conventional desktop window systems to embrace windows for images, audio streams, video streams, etc. (e.g., see [68]). The fundamental extension is the incorporation of a more diverse set of media into the desktop environment than existed in first generation window systems. Another class of electronic meeting rooms employ a collection of geographically distributed physical rooms augmented with computer-based tools (e.g, see [69, 70]. Participants assemble in the individual meeting rooms, and are logically gathered across the physical meeting rooms by the computer and network facilities. The emphasis in these projects is to provide effective means for extending normal "meeting tools" across the physically distributed rooms.

Virtual environment systems such as Dive [71], Rapport, VMM, and Archways [72], and RING [73] create another level of immersion into the work by virtualizing a shared room and its artifacts, *including its participants.* A participant has a virtualization that is placed in the environment where it interacts directly with shared objects, including other participants. There is considerable contemporary literature describing virtual environments to support electronic meetings [74, 75, 76, 77, 78].

Office workers usually build up a social model with other workers that they interact with while performing "company business" on the telephone. Frequently, the participants never actually meet one another in a face-to-face situation, yet they develop a social model that provides informal information exchange and procedure processing. Our conjecture is that the amount of logical immersion provided by the system correlates with the way the system supports the social interactions among the participants. An important rationale for introducing virtual environments into the collaborative system is to provide broad bandwidth facilities for informal communication, and as an environment to support a rich social model among the collaborators. Our conjecture is consistent with the rationale for any electronic meeting room or virtual environment study.

An Experimental Model-Based Virtual Environment. The *Model-Based Virtual Environment* (MBVE) is a virtual environment to support extended ICNs [79, 80]. The

rationale for extended ICNs is described in Section 4; we can summarize our approach by noting that besides the addition of goal nodes, we expect the language to identify regions in the 3-space model of Figure 7 in which the corresponding steps should be executed. The basis of the workflow system architecture is the same as described in Section 3. However, each user can enter the MBVE at any time to explore the model, to establish a tailored contextual assistance profile, or to discuss workcases and step interpretations with other team members. In cases where there is low detail, low conformance, declarational domains, there will be a need for the team members to discuss and negotiate the way the work is to be performed. The MBVE provides a high bandwidth environment in which unstructured communication can occur, yet where the workflow model is a fundamental part of the environment. The procedure model can be used to "see how things are supposed to be done" when exceptions arise.

The intangible contribution of the MBVE is expected to be its support of social models. We have overwhelming anecdotal evidence that these social models are absolutely necessary in group work, yet we have no quantitative metrics of the models nor how to support them. Therefore, we have chosen a direction that eliminates various barriers to informal communication, since we believe that informal communication is the fundamental enabler for supporting the social model.

The cost implementing the the MBVE is high (we have been working on our prototype for about a year and a half, yet it is still is not at a level suitable for end user use). The distributed system must support real time multimedia sessions among group members' computers. Further, the modeling system must be integrated into the virtual environment. Based on the rationale we have described in this paper, we believe it is necessary to incorporate a capability of this nature, for a workflow system to replace a manual system (with its inherent social model).

5.2. Computation Support

The level of domain-specific computation support is an important criterion in characterizing a CSCW system. The y-axis orders the amount of computation in terms of the degree to which the system "automates" computation; this ordering is admittedly intuitive.

General purpose computer hardware is intended to be used in all information processing applications — from accounting to missile tracking. The software specializes the system's behavior to focus on a particular domain or problem. Thus the first level of computation support is the presence of generic, high level programming languages for the system. This allows professional programmers to define the specific behavior of the hardware so it provides a useful function to the end user.

Domain-specific programming languages are intended to provide higher level of

computational support by enabling experts in a domain to apply their knowledge to programming without becoming programming specialists. Thus, in our characterization, these languages provide a higher level of domain-specific computation support than do generic programming languages.

Personal productivity tools are typically horizontal applications that can be applied to many different domains. These tools include spreadsheets, web page editors, desktop publishing software, etc. We place these tools are higher on the computation support axis since they enable larger groups of users to use the computer for their work than is possible with domain-specific ("end user") programming languages.

The next level of specialization is vertical application software, or software written to solve a specific task. This includes a program to generate the monthly accounts receivable and payable, to compute the orbit of a satellite, or to simulate war games. This level of support is the highest, since it allows the user to perform some set of commands to exercise the program's behavior.

The final level of computation support in the figure is computation support through system facilities for contextual assistance. Our Bramble contextual assistance system provides support for all levels of computational assistance.

Contextual Assistance. The presence of goal nodes in an ICN suggests the need for a system mechanism to integrate human activity with the operational activity in the normal workflow model. When a task with unstructured work is delegated to a human actor, the request is made from a specific contextual environment in which the task is to be performed. The system ought to enable the actor to determine details of the environment from which the task was delegated. The task specification will contain some form of a declarational specification for the task which can be given to the actor when the task is delegated; however, the workflow system should be prepared to provide additional information (regarding the workcase, step environment, or workflow model) that may exist in the system should the actor desire it. The nature of such information can vary from simple to complex: we characterize types of contextual information in 5 levels [81]:

- **Static Information.** The coordination model implicitly contains information that can be useful to the human actor, e.g., it can provide a view of the coordination model so that the actor can see what should have already been done. The model should provide a mechanism by which the actor can inspect the coordination model and other operationally specified tasks.
- **Scripted Information.** The workflow designer can anticipate information that could be useful to the actor — typically referred to as "help" information — and incorporate it into the goal node description.

- **Dynamic Information.** Some information in the workflow system depends on the way the workcase was processed, i.e., it depends on the details of the workcase and the state of the system. For example, the actor may wish to know which specific actor performed some specific task for the workcase. Support to enable the actor to obtain this type of information will have to be provided by the system, but the model must be consistent with this usage.
- **Computed Information.** If an actor is to be repetitively executing an unstructured task, the workflow system can provide facilities to enable the actor to establish a profile for contextual information that he or she will require to execute the given task. That is, the model must enable the actor to provide an operational specification to the workflow system that defines how the workflow system should supply contextual information to the actor at the time that an unstructured task is assigned.
- **Inferred Information.** The workflow system can be made to be arbitrarily complex, e.g., it may contain a knowledge base and inferencing engine that provide very complex contextual information to the actor (as do learning systems, expert systems, etc.). This level of support essentially causes the model to cross over into the domain of declarative models.

Workflow systems do not typically allow such unstructured work to appear explicitly as a task in the computation model (although manual operations are embedded in many operational task specifications). The contextual assistance facility explicitly supports models and systems that allow unstructured work to be a first class part of the model, and to have specific support for the approach. The Bramble contextual assistance systems allows an analyst to design workflow system that support static, scripted, dynamic, and computes information [81].

CSCW researchers generally favor higher levels of computer support for collaborative work. While our placement of contextual assistance on the computation axis may be arguable, the notion of increasing levels of support for computation is well-accepted. As a rule of thumb, the higher the level of support (within a domain), the better users like the system.

5.3. Work Coordination Support

The z-axis is the degree with which the system incorporates workflow scheduling into the environment. CSCW systems embracing the situated work philosophy attempt to design the system so it creates a electronic environment that is perceived to be similar to (or at least only incrementally different from) an extant manual environment. Leverage is gained from computers through computation support and incremental "micro level automation" by which various tasks in the environment are accomplished using software

tools [12]. This fits well with the trend in personal productivity tool development.

In general, the z-axis of Figure 8 represents the nature of automation of the coordination model in a CSCW system. Situated work tends to minimize the importance of automatic coordination, so we place it near the origin. Manual coordination refers to systems providing tools to enable the group to coordinate their activities, including shared memory, shared resources, and synchronization mechanism such as locks.

In contrast to the situated work proponents, Malone and Crowston argue that computers should explicitly be used to coordinate tasks [15]. Workflow models are one approach for doing this, though there are other models for accomplishing this task. In our characterization, we differentiate between workflow system that provide *suggestions* of work to be performed by human users versus systems that automatically schedule work for the human worker (such as FlowPATH [49]).

FlowPATH provides another form of flexibility for dealing with coordination. The *sendTo* operation allows the system's users to violate the coordination model at times when that was allowed (regions in which the conformance parameter is low), and in which it makes sense [4] (InConcert provides a similar ability [29]). These ideas can be extended with models such as goal-based ICNs, contextual assistance, and suitable virtual environment.

People vary in how they would like a CSCW system to behave in terms of this characteristic. A flexible CSCW system should allow its designers and users to control the point along the axis at which roles are executed. This provides the primary justification for incorporating workflow models into a virtual environment: the presence of the procedure model provides the environment's users with an artifact to represent and discuss the work they are jointly trying to accomplish. However, this presence does not preclude informal communication; it establishes a context in which the office work can be enacted. The amount of conformance to the model can be controlled; the model can document and assist in collaborative work, or it can explicitly schedule step execution within the distributed system.

6. Conclusion

Workflow technology is now nearly twenty years old. In the 1970s it showed promise as an important new technology for office information systems. In the 1980s, the technology hit the realities of enactment: operational models were too rigid and prescriptive to provide computational support in an environment where humans had to work with the computers to accomplish collaborative work. This led to a displacement of the technology until the 1990s.

Today, workflow technology once again has a relatively large following, though the researchers in the area are now very aware of the interdisciplinary nature of

successful system. The successful workflow system must extend the models to include declarational specifications along side operational ones. The systems must represent both structured and unstructured work within a procedure. Further, the models and systems must support varying degrees of conformance to the coordination model specification, depending on the situation. The goal of leading edge systems is to support these extended workflow enactment languages in a collaboration environment supporting high logical immersion into the environment, with high computation support, and variable coordination support.

The MBVE is an example of how a workflow system can be explicitly supplemented with mechanisms to support informal communication. In addition to robust informal communication support, it provides facilities for saving, organizing, and retrieving conclusions from such informal communication. In particular, such systems augment the workflow environment with mechanisms to assist a group of humans to collaboratively and informally study, design, and evolve the formal model. The MBVE is a domain-specific, multiperson, virtual meeting room to support collaborative browsing, navigating, querying, and updating of the information. A basic modeling system establishes a well-defined design domain, but the overall system must provide additional facilities to enable a group of designers to immerse themselves in the model *and* the informal information to discover problems in the design, to negotiate different approaches to resolution, and to review the design rationale for various model components. Normally this is done through conversation and meetings. The MBVE creates a CSCW environment to support informal conversation and more formal meetings among a group of geographically dispersed designers where the context of the meeting is set by the workflow model and its associated information base.

We observe that if any new system removes or changes an organization's underlying social model, it will probably only be effective if the users are willing to adapt their fundamental social behavior to fit the system's implied model. In this research, our premise is that it is risky to build computer systems depending on social change for their success. Instead, we advocate collaboration systems based on formal coordination models, yet which explicitly support existing social models in the organization through informal broadband communication. We believe this is accomplished by having a high degree of logical immersion, a relatively high level of computation support, and a moderate and end-user-controlled amount of automatic coordination.

Acknowledgments

This work was supported under NSF Grant number IRI-9307619. The author also wishes to acknowledge the years of collaboration on workflow-based modeling and systems with Skip Ellis, first at Xerox PARC and now at the University of Colorado. Rick Blumenthal

is the lead researcher on the Bramble contextual assistance effort.

References

[1] C. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, January 1991.

[2] Jonathan Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 37(1):93–105, January 1994.

[3] Special Issue of IEEE Computer on Computer-Supported Cooperative Work, May 1994. James D. Palmer and N. Ann Fields, Guest Editor.

[4] Clarence A. Ellis and Gary J. Nutt. Modeling and enactment of workflow systems. In *Advances in Petri Nets 93*, pages 1–16, June 1993. Invited paper.

[5] Dimitrios Georgakopoulos, Mark Hornick, and Amith Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):117–153, April 1995.

[6] Jintae Lee, Gregg Yost, and the PIF Working Group. The PIF process interchange format and framework. Technical report, University of Hawaii Information and Computer Science Department, February 1996.

[7] Thomas W. Malone, Kevin Crowston, Jintae Lee, and Brian Pentland. Tools for inventing organizations: Toward a handbook of organizational processes. In *Workflow 95 Conference Proceedings*, pages 57–78, 1995. Also appears in Proceedings of 2nd IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises, April, 1993, and as MIT Sloan School technical report CCS WP No. 141, Sloan School WP No. 3562-93.

[8] Ronni T. Marshak. Characteristics of a workflow system. *Workgroup Computing Report*, 16(5):2–3, May 1993.

[9] B. Silver. Bis guide to workflow software. Technical report, BIS Strategic Decisions, September 1995.

[10] Keith D. Swenson and Kent Irwin. Workflow technology: Tradeoffs for business process re-engineering. In *Conference on Organizational Computing Systems*, pages 22–29, 1995.

[11] WFMC Members. A workflow management coalition specification: Glossary and document of understanding. Technical Report Document Number TC00-0011, Workflow Management Coalition, Brussells, Belgium, August 1994.

[12] Lucy A. Suchman. Office procedure as practical action: Models of work and system design. *ACM Transactions on Office Information Systems*, 1(4):320–328, October 1983.

[13] Special Issue of Communications of the ACM, September 1995. Lucy Suchman, Guest Editor.

[14] Mike Robinson and Liam Bannon. Questioning representations. In *Proceedings of the Second European Conference on Computer-Supported Cooperative Work — ECSCW ́91*, pages 219–233. Kluwer Academic Publications, 1991.

[15] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.

[16] M. Hammer, W. G. Howe, V. J. Kruskal, and I. Wladawsky. A very high level programming language for data processing applications. *Communications of the ACM*, 20(11):832–840, November 1977.

[17] Michael D. Zisman. *Representation, Specification, and Automation of Office Procedures.* PhD thesis, Wharton School, University of Pennsylvania, 1977.

[18] Clarence A. Ellis and Gary J. Nutt. Office information systems and computer science. *ACM Computing Surveys*, 12(1):27–60, March 1980.

[19] Jonathan Grudin. Why CSCW applications fail. In *Proceedings of the CSCW88 Conference*, 1988.

[20] Esther Dyson. Workflow. Technical report, EDventure Holdings, September 1992.

[21] C. Mohan. Tutorial: State of the art in workflow management system research and products. a tutorial at the ACM SIGMOD International Conference on Management of Data, June 1996.

[22] Gary J. Nutt. Using workflow in contemporary IS applications. Technical Report CU-CS-663-93, Department of Computer Science - University of Colorado, Boulder, August 1993.

[23] Shi-Kuo Chang, Tadao Ichikawa, and Panos A. Ligomenides. *Visual Languages.* Plenum Press, 1986.

[24] Special Issue of IEEE Computer on Visualization in Computing, October 1989. Allen L. Ambler and Margaret M. Burnett, Guest Editors.

[25] Steven Tanimoto, editor. *Proceedings of the 1992 IEEE Workshop on Visual Languages.* IEEE, September 1992.

[26] Alexander Forst, Eva Kűhn, and Omran Bukhres. General purpose work flow languages. *Distributed and Parallel Databases*, 3(2):187–218, April 1995.

[27] P. David Stotts and Richard Furata. αTrellis: A system for writing and browsing petri-net-based hypertext. In G. Rozenburg, editor, *Advances in Petri Nets '90*, pages 471–490. Springer Verlag, 1990.

[28] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2), 1994.

[29] Inconcert 3.0: Adaptive workflow software for continuous process improvement, August 1996.

[30] Shirish S. Hardikar. Analyzing your processes before automating. In *Workflow 95 Conference Proceedings*, pages 91–108, 1995.

[31] P. Zave and W. Schell. Salient features fof an executable specification language and its environment. *IEEE Transactions on Software Engineering*, SE-12:312–325, February 1986.

[32] W. Bruce Croft and Lawrence S. Lefkowitz. Task support in an office system. *ACM Transactions on Office Information Systems*, 2(3):197–212, 1984.

[33] Kum-Yew Lai, Thomas W. Malone, and Keh-Chiang Yu. Object lens: A 'spreadsheet' for cooperative work. *ACM Transactions on Office Information Systems*, 6(4):332–353, October 1988.

[34] M. Broy. *Control Flow and Data Flow: Concepts of Distributed Programming.* Springer Verlag, 1985.

[35] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[36] James L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Inc., 1981.

[37] Jeffrey D. McWhirter and Gary J. Nutt. Escalante: An environment for the rapid construction of visual language applications. In *Proceedings of the 1994 Symposium on Visual Languages*, pages 15–22, October 1994.

[38] Gary J. Nutt. An experimental distributed modeling system. *ACM Transactions on Office Information Systems*, 1(2):117–142, April 1983.

[39] Gary J. Nutt and Paul A. Ricci. Quinault: An office environment simulator. *IEEE Computer*,

14(5):41–57, MAY 1981.

[40] Jacques Bicard-Mandel, Martin Ader, and Josep Monguio. COP: A Petri net based coordination mechanism for scheduling production workflows. In *Computer-Supported Cooperative Work, Petri Nets and Related Formalisms (a workshop at Petri Nets 93)*, pages 42–55, 1993.

[41] Gary J. Nutt. *The Formulation and Application of Evaluation Nets.* PhD thesis, University of Washington, 1972.

[42] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use, Volume 1.* Springer-Verlag, 1992.

[43] Raul Medina-Mora, Terry Winograd, Rodrigo Flores, and Fernando Flores. The action workflow approach to workflow management technology. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 281–297. ACM, 1992.

[44] Filenet visual workflo software, March 1996.

[45] Clarence A. Ellis. Information control nets: A mathematical model of office information flow. In *Proceedings of the 1979 ACM Conference on Simulation, Measurement and Modeling of Computer Systems*, 1979.

[46] C. A. Ellis and P. A. Morris. The information control nets model. *Performance Evaluation Review*, 8(3), November 1979.

[47] Gary J. Nutt. A Simulation System Architecture for Graph Models. In G. Rozenburg, editor, *Advances in Petri Nets '90*, pages 417–435. Springer Verlag, 1990.

[48] Gary J. Nutt, A. Beguelin, I. Demeure, S. Elliott, J. McWhirter, and B. Sanders. Olympus: An Interactive Simulation System. In *1989 Winter Simulation Conference*, pages 601–611, Washington, D.C., December 1989.

[49] Bull S. A. *Introduction to FlowPATH*, May 1992. Manual No. 44 A2 60XM.

[50] David A. Edwards and Martin S. McKendry. Exploiting read-mostly workloads in the filenet file system. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 58–70, 1989.

[51] Ronni T. Marshak. Action technologies workflow products. *Workgroup Computing Report*, 16(5):4–11, May 1993.

[52] Fernando Flores, Michael Graves, Brad Hartfield, and Terry Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, April 1988.

[53] D. Harel. Statecharts: a Visual Formalism for Complex Systems. *Science of Computer Programming - North-Holland*, 8:231–274, 1987.

[54] Peter Newton and James C. Browne. The code 2.0 graphical programming programming language. In *Proceedings of the ACM Conference on Supercomputing.* ACM, July, 1992.

[55] Narayanan Krishnakumar and Amith Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2):155–186, April 1995.

[56] D. R. McCarthy and S. K. Sarin. Workflow and transactions in inconcert. *IEEE Data Engineering*, 16(2):53–56, June 1993.

[57] B. Karbe, N. Rampserger, and P. Weiss. Support of cooperative work by electronic circulation folders. In *Proceedings of the ACM COIS '90*, pages 109–117, 1990.

[58] Thomas Kreifelts, Elke Hinrichs, Karl-Heinz Klein, Peter Seufferet, and Gerd Woetzel. Experiences with the DOMINO office procedure system. In *Proceedings of the Second European Conference*

on Computer-Supported Cooperative Work — ECSCW ́91, pages 117–130. Kluwer Academic Publications, 1991.

[59] Thomas W. Malone, Kevin Crowston, Jintae Lee, and Brian Pentland. Tools for inventing organizations: Toward a handbook of organizational processes. Technical Report CCS WP No. 141, Sloan School WP No. 3562-93, M.I.T., Sloan School of Mangement, 1993.

[60] Ronald M. Baecker. *Readings in Groupware and Computer-Supported Cooperative Work*. Morgan Kaufmann Publishers, Inc., 1993.

[61] Heikki Saastamoinen, Markku Markkanen, and Vesa Savolainen. Survey on exceptions in office information systems. Technical Report CU-CS-712-95, Department of Computer Science - University of Colorado, Boulder, 1994.

[62] Kenneth L. Kraemer and John Leslie King. Computer-based systems for cooperative work and group decision making. *ACM Computing Surveys*, 20(2):115–146, June 1988.

[63] Lawrence S. Lefkowitz and W. Bruce Croft. Planning and execution of tasks in cooperative work environments. In *IEEE AI Conference*, pages 256–261, 1989.

[64] Patricia Sachs. Transforming work: Collaboration, learning, and design. *Communications of the ACM*, 38(9):36–44, September 1995.

[65] Clarence A. Ellis and Najah Naffah. *Design of Office Information Systems*. Springer-Verlag, 1987.

[66] Clarence A. Ellis and Jacques Wainer. Goal based models of collaboration. *Collaborative Computing*, 1:61–86, 1994.

[67] Clarence A. Ellis and Gary J. Nutt. The modeling and analysis of coordination systems. In *ACM 1992 Conference on Computer-Supported Cooperative Work*, 1992. workshop position paper.

[68] Ian E. Smith, Scott E. Hudson, Elizabeth D. Mynatt, and John R. Selbie. Applying cryptographic techniques to problems in media space security. In *Proceedings of ACM 1995 Conference on Organizational Computing Systems*, pages 190–196. ACM, 1995.

[69] Clarence A. Ellis and et al. Project nick: Meeting analysis and augmentation. *ACM Transactions on Office Information Systems*, 5(2), April 1987.

[70] J. F. Nunamaker, Alan R. Dennis, Joseph S. Valacich, Douglas R. Vogel, and Joey F. George. Electronic meeting systems to support group work. *Communications of the ACM*, 34(7):40–61, July 1991.

[71] Lennart E. Fahlen, Charles Grant Brown, Olov Stahl, and Christer Carlsson. A space based model for user interaction in shared synthetic environments. In *Proceedings of Interchi '93*, pages 43–48, April 1993.

[72] David A. Berkley and J. Robert Ensor. Multimedia research platforms. *AT&T Technical Journal*, 74(5):34–45, September/October 1995.

[73] Thomas A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *1995 Symposium on Interactive 3D Graphics*, pages 85–92. ACM, 1995.

[74] Special Issue of AT&T Technical Journal on Multimedia, September/October 1995. Nikil Jayant, Technical Reviewing Editor.

[75] Duane K. Boman. International survey: Virtual environment research. *IEEE Computer*, 28(6):57–65, June 1995.

[76] Domenica Ferrari, editor. *Proceedings of the Second ACM International Conference on Multimedia*. ACM, 1994.

[77] Special Issue of IEEE Computer on Virtual Environments, July 1995. David R. Pratt, Michael Zyda, and Kristen Kelleher.

[78] Special Issue of IEEE Computer on Multimedia Systems and Applications, May 1995. Arturo A. Rodriguez and Lawrence A. Rowe, Guest Editors.

[79] Gary J. Nutt. Model-based virtual environments for collaboration. Technical Report CU-CS-799-95, Department of Computer Science, University of Colorado, Boulder, December 1995.

[80] Gary J. Nutt, Joe Antell, Scott Brandt, Chris Gantz, Adam Griff, and Jim Mankovich. Software support for a virtual planning room. Technical Report CU-CS-800-95, Department of Computer Science, University of Colorado, Boulder, December 1995.

[81] Richard Blumenthal and Gary J. Nutt. Supporting unstructured workflow activities in the Bramble ICN system. In *Proceedings of ACM 1995 Conference on Organizational Computing Systems*, pages 130–137, August 1995.