BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-91-M18

"Form-Centered Workflow Automation Using an Agent Framework"

by

George Fitzmaurice

Form-Centered Workflow Automation Using an Agent Framework

by

George Fitzmaurice

ScB, Massachusetts Institute of Technology, 1988

Department of Computer Science

Brown University

Thesis

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in the

Department of Computer Science at Brown University

August, 1991

Table of Contents

This thesis by George Fitzmaurice is accepted in its present form by the Department of Computer Science as satisfying the thesis requirement for the degree of Master of Science.

Date Aug 28, 1991

[illegible]

Prof. Andries van Dam
Advisor

Acknowledgments

This thesis is a culmination of many people's efforts who aided in many facets of the thesis ranging from conceptual brainstorming, analysis, design, and most importantly, moral support.

Professor Andy van Dam was instrumental in ensuring the successful completion of this thesis. The work described in this document stems from many detailed discussions with Andy who contributed significant ideas and analysis, unlimited energy and managerial skills that kept the project on track and inspired me to work the hardest I have ever worked. Andy, always working on ten or more things at the same time and at a whirlwind pace, was always accessible in person, and I enjoyed many of the late night, long distance phone calls and fax exchanges. Thanks also to Lori and Tina who kept tabs on Andy which allowed me to keep in constant contact with Andy.

The folks at BLOC Development (Peter Preksto, Ron Frohoc, et.al.) provided great reviews of early design ideas and always kept me honest. Randy Foregaard and Eugene Lee of Beyond Inc. helped me get a quick understanding of the existing mail systems and technology trends. Karen Smith Catlin contributed an interesting perspective on workflow by providing me with a set of forms from Hitachi Europe Ltd.

Tennis provided me with an outlet to escape the work and stress and helped me to relax and cope with the matters at hand. Thanks to all of my tennis cohorts including Bern, Rob, Chris, Laura and Muru.

The research was sponsored in part by IRIS and I thank them for providing the working environment and computers for the project. Marty Michel extended the project and supplied valuable technical information and analysis. Thanks to Rose Antoni for her constant encouragement and administrative wonders. Norman Meyrowitz, who supported the original agent research, served as my initial advisor and helped me investigate and evaluate masters topics; I will always be impressed with his technical insight. Also, the workflow prototype would not be possible without thanking Muru Palaniappan for co-designing and implementing the original agent framework with me. A special thanks to all at IRIS who helped me to juggle classes and a full-time job with astonishing success.

I am amazed at the amount of work that was undertaken in the less than three month lifespan of this thesis. To my close friends, I described the experience as a PhD bootcamp and on reflection I realize that I not only learned about my research topic but also how to conduct high quality research.

This thesis is dedicated to my sisters (Jean, Julie, Elizabeth) and my parents (mom and dad) who supplied enormous emotional support and encouragement throughout the entire grueling process. Thank you all.

— G.F.

1 Overview and Philosophy

1.1 Introduction

Office workflow can encompass many different disciplines (e.g., psychology, group dynamics, transactional analysis, queuing theory) and technologies (e.g., facsimile machines, computers, applications, networks). We have taken a much more restricted view of workflow as deterministic, centrally designed and form-centered workflow. A forms-centered approach means that at each workflow stage, some processing occurs that usually involves the form. The forms are live; users can fill in fields of a form, launch additional applications and communicate with documents produced by those applications (e.g., spreadsheets).

Paper forms are used extensively in many organizations and often serve to record events, request supplies or services, and notify people. The forms serve to encode the procedure that is required to reach a desired organizational goal.

We use the term workflow to describe the entire process that the forms undergo in an organization to reach a desired goal. Many organizations have strict procedures for servicing a request which involves one or more forms at any given stage in the procedure. At each stage, workers typically interact with forms, examining, filling-in, and taking actions based on the data present on the forms. The forms travel between workers, departments, and organizations that usually are not geographically co-located. Enterprise-wide workflow encompasses forms that are used throughout a large organization having multiple LANS (and WANS), hosts, and a variety of workstations.

In our view, electronic forms need to be intelligent, active, and dynamic objects that interact with one another and the environment in which they exist. Many early workflow systems have a restricted design which treats forms as independent units of processing which expect each form to work in isolation. This is not true in our approach. We consider workflows that span a form set and the forms can communicate and interact with one another. Routing depends on the form set, not just individual forms.

The computer industry proposes to automate the workflow process by supplying electronic forms[1], intelligent form routing, security, user notification, resource balancing, and

[1]Future references to the term "form" mean a form instance unless otherwise specified.

tracking applications for users and managers to design, implement and monitor the workflow process. Each of these components will be described and discussed later; however, intelligent form routing will be emphasized in this document and in our design.

1.2 Centrally Designed vs. Centrally Controlled routing

In analyzing and designing workflow systems, it is crucially important to understand the differences between centrally designed and centrally controlled routing. The model we have defined concentrates on deterministic, form-centered workflow routing which is centrally designed and defined before the form in initially used. Alternatively, an ad-hoc routing system would rely on each form-receiving user to manually decide and advance the form to the next user. This approach is prone to errors and, consequently, we have adopted a limited notion of ad hoc routing by allowing users to manually copy a form and send it to a co-worker.

Centrally controlled routing addresses the question of where the routing information and definition are kept. There are three major design types: (1) the routing script is attached to the form and travels with the form; this is the decentralized approach (2) routing information is managed in a central *formbase* (form database) for all form types, (3) each form type has a private formbase to access routing information. Centrally controlled routing also identifies whether a single component or process is responsible for executing the routing script. There are three major approaches: (1) the routing script is interpreted locally at each user's site, (2) a central server process all of the routing requests, (3) a different server exists to process routing requests for each form type.

In addition to these routing classifications, when we consider workflows that span multiple forms, system designers need to decide whether there will be one routing script for all of the forms or individual scripts for every form. In the Background section we will pay particular attention to which routing scheme was chosen and identify the benefits and detriments of each approach.

1.3 Working Assumptions

It is believed that enterprise-wide workflow may involve on the order of tens of thousands of different form types and thousands of workflow schemas for large organization such as Chrysler and Boeing. While enterprise-wide workflow needs to be able to operate in a heterogeneous, distributed environment, we deal here only with workflows that span groups of user working on a single local area network such as might be found in a small business or a department of a larger organization.

This model supports a division of labor based on the principle of locality of reference. That is, it is expected that local departments of large organizations do not make use of all of the form classes that the organization has defined. In reality, they use only a fraction of the total number of form classes. Similarly, a department will most likely use a small set of workflow schemas in their everyday work. Having a centralized form database is impractical. A large organization would place tremendous stress on a single database process trying to service its community.

The model assumes that there are several hundred form types and dozens of workflow schemas defined. It is not clear what modifications to our design will be necessary to support large, enterprise-wide workflow environments. We are trying to be sensitive to the needs to have our solution scale-up, but more analysis needs to be done to determine whether that is feasible.

1.4 Example of Workflow

In order to understand what types of functionality and information needs to be provided in a workflow automation system, a simple workflow schema is described. One particular company requires that its workers submit an overseas travel application before conducting a business trip overseas. The form requests that the applicant fill in their name, trip number, purpose of the visit, a schedule with sites, people, dates and preferred accommodations. Once the form is filled in, the applicant's manager must approve of the form. Next it travels to a general manger who authorizes the request. The form is then used to initiate the travel arrangements for the applicant.

In this workflow schema, there are four distinct stages: (1) user fills in form to make a request, (2) manager approval, (3) general manager approval, (4) initiate travel arrangements. Consequently, the form is expected to undergo four routing transferrals, the first of which entails instantiating a new form instance.

While filling out the form, the user can be aided by having some of the fields be filled in before (or as part of) opening the form. For example, the applicant's name and other identification information can automatically be filled in from the user's ID and the user's profile in a personal database. Field values such as the trip number can be computed as well. In addition, the form validates the data while the user fills in each field of the form.

Once the user submits the request, the next recipient needs to be determined and sent the form. If the request is not approved, the form must be routed back to the applicant who must be notified.

The workflow designers want to selectively enable access to workflow status and tracking information. Administrators or an individual may be enabled to inquire about certain kinds of status information. The types of information which may be available are status information about what stage the workflow is currently in and who is working on the request.

Managers would like to examine summary statistics on the workflow schema such as: how many overseas travel applications were submitted, how many were approved, what is the average time an application takes to be processed, which month had the most application request. The accounting department may want to collect statistics on the average cost of the trips and the amount spent on trips by individual workers or departments.

1.5 Workflow Functionality

The simple workflow example above highlights the basic functionality needed in a workflow system. However, more progressive concepts need to be provided in order to mimic and improve on the current paper-based workflow functionality. These concepts are described below.

Terminology

Before continuing, it is necessary to understand some frequently used terminology. A *workflow schema* dictates which forms and in which order the forms (and their partitions) are filled in and routed to users and also what external processing takes place at each hop (i.e., launching an application and importing form data). While the workflow schema provides the definition and description of an office workflow procedure, a *workflow instance* holds the data and state information for a workflow procedure that is in progress or completed. Forms, in our model, are implemented as object-oriented classes and each *form class* defines the content (i.e., fields), presentation (i.e., the layout of the fields), behavior (i.e., actions associated with the fields such as verification) and routing (i.e., the ordering of who should process the form). Note that routing information for forms in a form set may largely overlap. A *form instance* holds the data and state information for a given invocation of the form class definition.

Form Sets & Partitions

Many workflow stages require the user to access and fill-in multiple forms. Therefore, the notion of having a workflow schema attached to a single form is impractical. Workflow schemas make use of one or more form sets. A *form set* is a collection of one or more logically related forms that can be accessed and operated on for a given workflow schema. Figure 1 shows three forms belonging to a form set. The forms are divided into regions, called *partitions*, that contain collections of logically related fields. Note that forms in a form set do not necessarily travel to all of the workflow steps.

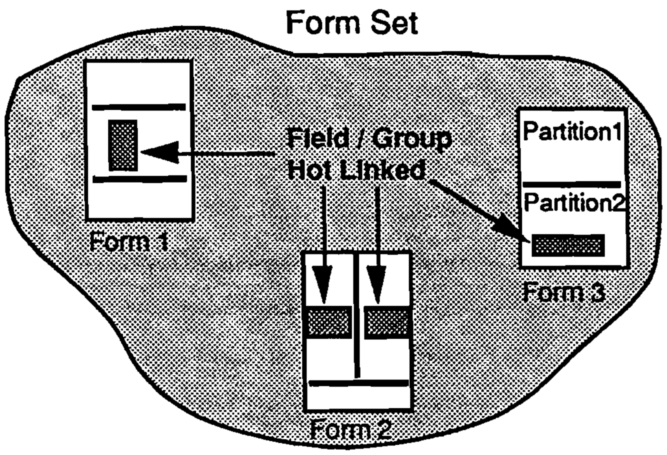


Figure 1: A Form set containing three forms. Each form is divided into two or three partitions of logically related fields. The forms contain a group of hot linked data fields.

Hot-Linking

Data will often be shared among forms within a workflow schema. For example, the initiator's name, address, title and social security number or other identifying numbers should be on many of the workflow forms. Mechanisms must exist to support hot-linking of data fields between forms. This allows all forms to receive immediate updates when any one of the hot-linked fields changes. Fields can be grouped and groups can be hot linked as well; neither individual fields nor groups need be displayed in the same way on different forms. Note that we associate linking with a workflow, not with the form definitions themselves.

Form Membership

Forms can belong to many workflow schemas. For example, a product price list form could belong to an inventory workflow and a purchasing workflow schema. When the price of an item changes, both forms, their form set and their workflows are affected and need to know of the change.

Parallel Routing

A workflow stage is a stopping point in which users, typically, need to perform some action before advancing to the next stage. Each stage of the workflow does not necessarily need to follow a linear pattern. Often a form is divided into regions (i.e., partitions) which workers can process in parallel since the fields to be filled out are mutually exclusive (they may share an arbitrary amount of read-only information).

Split/Merge of a form

When a form is being worked on in parallel, the form is considered *split* between one or more users. Once users finish processing the form and submit it, the data must be reconciled and merged.

Cyclic Routing

In addition to the parallel routing, workflows may be defined in which a routing cycle occurs. For example, a review process or a negotiation may require that a form or form set travel back and forth to the same users until a condition is met which breaks the need to cycle. Note that cycles should not be restricted to two users only one hop away but can include many users and hops for one complete workflow cycle.

Roles

For a workflow schema to remain generic, roles must be used instead of specifying exact userIDs. The form routing logic will want to indicate a role (e.g., Manager, Department Head) when a form needs to be forwarded to the next workflow stage.

Notification

Users should be notified of new workflow tasks that they have been assigned to process. In fact, the workflow requests can be prioritized for users, allowing them to determine which requests are most critical. The workflow requests have an associated deadline and users can be warned when the deadline is approaching or can be notified that the request is overdue.

Customized Views

The electronic forms need to be active and adaptive depending on the workflow stage and current user. Forms can be tailored in terms of their visual presentation, layout, content, and security. For example, a worker may only want to see or be allowed to see the fields that s/he uses to process the form at the current workflow stage. Moreover, the worker might be allowed to exercise his or her own viewing preferences to alter the layout and appearance of the form when displayed or the workflow designer might tailor the appearance to the skill level or each class of workers. Certain forms may contain sensitive data which can be made invisible at workflow stages. Fields within a form may have read and write access controls to prevent users from filling-in certain fields (i.e., a user is prevented from filling-in the manager's authorization field).

Tracking
Users need to have tools that allow them to see the current state of workflows which are in progress. In addition, tools that help users manage their work requests (e.g., prioritize or sort them) are very important in a successful system. Note that the tracking data could probably be used by a variety of applications.

Collective Tracking & Resource Balancing
Statistical tracking data that summarizes the collective workflow jobs can also be computed to aid designers, for example, in determining the workflow bottlenecks. The tracking data can also aid managers in identifying trouble spots in the workflow route. Similar to a network manager which monitors the throughput of gateways or servers, workflow components can monitor if there is a backup of requests or identify which steps take the longest to complete. The data could also be used to dynamically adapt the workflow by re-routing requests through less congested servers or to workers who have a smaller work queue. In short, a sophisticated workflow system will be able to perform resource balancing to obtain maximum throughput.

Authentication
The workflow system needs to prevent unauthorized access to form and workflow schema definitions as well as data belonging to form instances and workflow instances. Access and concurrency control at the field level of a form is necessary to support parallel routing.

Atomic Transactions
The model should be robust enough to support a rich set of workflow transactions. That is, we should be able to define atomic transactions that have the same functionality of databases (e.g., can be journaled, and can recover from a system failure by rolling back to a consistent state).

All of this functionality needs to be provided by a state-of-the-art workflow system. It is proposed that an agent-based system could serve as the core building block for a workflow framework. Therefore, we describe a model that builds upon an existing agent framework, Brown's Envoy System [45], and examine the feasibility of the model to support automated office workflow.

1.6 Overview of Our Model

In designing our workflow system, our goal was to support all of the above mentioned functionality. While we are initially concentrating on the intelligent routing aspects of workflow, the model can easily be extended to incorporate many of the features.

The decision to use Brown's agent framework[] as the core for our workflow model was based on two factors: (1) the agent framework already existed and its architecture was touted as being extensible; we wanted to test this theory, and (2) agents are a natural choice since they can easily handle decentralized processing. That is, the agent framework provides workflow functionality such as: notification mechanisms, automated scheduling and management of tasks, tracking, and resource locating.

The model dedicates one server process per form type. This server furnishes form-level requests for any form instances of its given type. It is responsible for field and form level access control and persistent storage of its forms. Users interact with an independent frontend process which receives a package of data from the server which allows the frontend to perform field-level transactions without communicating back to the server (except for hot-link data exchanges).

Routing logic is centrally aggregated on a form-type basis and the definition and state information for the routing code resides in the server. The routing code is executed only when a user submits its work request.

An additional process per workflow schema serves as a communication hub to the rest of the components. Its primary function is to manage and track information of workflow instances belonging to its schema.

1.7 Outline of Paper

First a lengthy review and analysis of existing systems that address the area of workflow automation is presented. During this analysis we determine how it satisfies and relates to our ideal workflow system. Next, we describe our agent-based workflow model and define a generic form routing library to be used in conjunction with a high level programming language (e.g., C++). This library provides an easy and familiar mechanism for workflow designers to program new workflow schemas. Our prototype and implementation are delineated next along with the user interface for our prototype. We end with an evaluation of our model, the lessons we have learned and talk about possible future work.

2 Background

This lengthy section first describes workflow systems in the research community and then commercial systems that are available today or coming to market shortly. The Logical Routing and the PAGES system are two of the most relevant research systems reviewed. Three of the systems known as Systems A, B, and C are under non-disclosure and only a brief description will be presented to highlight their approach. Because our design also spills over into agents and groupware, two short sections on these areas will give the reader pointers to related references.

2.1 Research Systems

2.1.1 Logical Routing

The Logical Routing project is a Message Management System (MMS) built at the University of Toronto [39]. The system addresses the need for logical routing of structured messages which can use information about themselves or about the system to affect their own processing. Mazer and Lochovsky define a routing specification language which supports: single message routing, parallel and cyclic routing, splitting and merging of messages. They have also incorporated some simple notification and the system is the only one reviewed that has attempted to address the issue of resource balancing. While this system is relatively old, it is one of the most relevant systems that deal with the problems of message routing. Their approach allows for routing to be centrally designed but is evaluated locally at each site. The system design, however, still does not consider a collection or set of messages; the messages travel and work mostly in isolation. See the appendix for a more detailed evaluation of this system.

First, it is important to realize that their system does not make many references to the notion of form automation. Instead, their office information system centers around message types which can be thought of as object classes (or, in our world, Form classes). A message is an instance of a message type. The messages are stored in a communication base which can be distributed or centrally located.

Each user of the communication base is modeled by an agent who plays a role (e.g., manager) within the MMS. The term agent is somewhat misused in this context. In the

MMS, an agent is simply a mapping mechanism and representation between specific userIDs and roles. Users may belong to more than one role and a role may include other roles.

The authors define three kinds of sites: *origin site* (the source of the message), *processing site* (intermediate sites), and *terminal site* (the last site in a route). Each site has an in-mail and out-mail tray to receive and send messages.

The routing specification of messages falls under two major categories: *type routing* (the message designer defines a route for all instances of the message) and *instance routing* (users define or modify the routing of a message instances that affects the present instance only). Overriding a message's route (i.e., ad hoc specification) can occur at both the type and instance routing.

At each site, originals or copies of a message instance can be sent to the next set of recipients. Copies are independent from the original in terms of content and the route they take after they are created. The system places a significant routing constraint when multiple originals are sent to a set of recipients during the next routing stage. All originals must evaluate the same routing specification and all must evaluate to the same next site (i.e., once the routing code is executed and the next site is determined, all originals will be forwarded to this site). Note that it is possible to route an original to sites when all, or some of, the recipients have processed the message.

An *automatic procedure* may be defined to block the routing of a message instance until some criteria are satisfied (e.g., waiting for another message to arrive or a field value). The authors, however, give no indication on how designers would define or install automatic procedures.

A routing specification for a message type consists of a single script definition which contains a set of subdefinitions for each potential site in the route. Time constraints on processing an instance may be specified for each site. Within a site specification, different routing code can be executed depending on how many times the given instance has visited the same site. If an error occurs during the routing process, designers can specify addition code for run-time exception handling (e.g., return message back to originator). Conditional routing constructs exist to allow messages to be forwarded to different sites based on supplied criteria being satisfied. Note that time constraints are checked upon an instance's arrival at a site and periodically during the message's stay at a site. The routing conditions are only checked during routing evaluation when the message has been processed. Finally, designers may specify whether the users of a given site can create instances of the same message type.

Some level of resource balancing is possible by specifying the following keywords during the routing forwarding command:

- FEWEST-MESSAGES <site 1> OR <site 2> OR <site N> – current instance is to go to which ever site currently has the fewest instances of the given message type.
- MOST-MESSAGES <site 1> OR <site 2> OR <site N> – go to which ever site has the greatest number of instances of the given type.
- LEAST-LOADED<site 1> OR <site 2> OR <site N> – go to which ever site has the least loaded CPU
- MOST-LOADED<site 1> OR <site 2> OR <site N> – current instance is to go to which ever site has the greatest loaded CPU

Implementation

The prototype system is implemented under UNIX and uses C and a relational database management system.

A central log provides a history of message instances, containing an entry for each creation, termination, and each movement of an instance from one site to the next. Database relations exist to support concurrent routing and authorizations.

A message type routing script is preprocessed (i.e., compiled) before invocations are possible. This phase deals with checking for proper syntax and expanding general site specifications and resolving roles.

Next, the message type definitions are distributed to sites. There are a few strategies for distributing: *member* (some message types are already known to all sites as part of the original system), *implicit* (all of the sites specified in the routing script will be sent the message type definition when the instance is created), *explicit* (the message designer can manually send the definition to a site), and *ad hoc* (a temporary copy of the message type is sent to a site; the copy is discarded once the message has been processed at the site).

Because the message type routing script is already specified as a collection of independent site definitions, the script can be partitioned on a site-by-site basis. Only the site specific script is propagated to the corresponding site (as opposed to the total routing specification script). The authors are not completely clear on when the script needs to travel with a message type or message instance. It appears that the routing script needs to travel to sites during at hoc routing and manual routing.

Three methods for evaluating the routing specification are described: *central evaluation* (a central authority, or process, evaluates the routing after each site completes processing the instance), *origin evaluation* (the originating workstation will make all the routing decisions), and *distributed evaluation* (after each site's processing is complete, the corresponding workstation evaluates the decomposed routing and sends the instance on to the next site). The distributed evaluation is used in the prototype.

The triggering of the routing evaluation at each site occurs when the user is finished processing the message instance. This occurs by the user placing the message in his or her out-tray. Automatic evaluation may occur, for example, if one of the time constraints have been triggered.

2.1.2 Electronic Circulation Folders

The ProMinanD system[29] is an Electronic Circulation Folders approach to office workflow developed by Karbe, Ramsperger, and Weiss. Their philosophy concludes that, in general, office work is full of exceptions, and that, in the long run, changes take place with respect to organization structure, assignments, office roles, and event tasks. They have also chosen a decentralized evaluation approach to workflow. The workflow scripts are centrally defined but the system promotes ad-hoc routing and dynamically modifying the routing procedures. There are no provisions to support parallel routing (nor split/merge) or hot-linking data between documents. The system is, however, one of the only designed to handle a collection of documents which are associated to a given workflow schema.

The model is based on a very traditional tool for processing an office task, the circulation folder (CF). Its contents consist of arbitrary but related documents which are to be worked on by office workers. Their addresses are written on the cover of the CF. An internal messenger service takes care of the transfer of closed CF's from an office worker's out-tray to the in-tray of the one whose address has been added to the CF's cover by the office worker at his discretion.

The electronic version of this model, ProMinanD, offers more functionality and greater efficiency over the paper version.

Users log into the system and select an office role that they will be performing. A graphical desktop gives the user access to office tools such as editors, and the ECF in-box and out-box.

Operations

There are operations "forward," "postpone," and "inform" which deal with the common migration of ECF's. The operation "forward" advances the ECF to the next office worker fulfilling the specified role. An office worker can postpone the processing of the work inside an ECF as well as issue the inform command to receive the status of an ECF (i.e., where the ECF has migrated from the worker's the out-box). Other operations on ECF include:

- *not me* – worker claims he is not the one who has to work on the current step. ECF is sent back to the previous worker because of "refusal."
- *refer back* – worker needs additional information
- *append* – add a migration step immediately following the worker's step. The idea here is that an office worker may want to have another office worker (normally a subordinate one) carry out or complement the current step.
- *delegate* – appends two steps after the current one. Whereas the first one is appended like before the second one comes back to the delegating office worker to get the results back and take over responsibility.
- *shortcut* – removes a step in the workflow
- *shift* – delays a step in the workflow to be worked on later (perhaps a worker who is responsible for the step will not be in for some time)
- *fetch back* – retrieve an ECF which has been moved to the next worker's in-box. Can only be retrieved if the next worker has not processed the ECF yet.
- *cancel ECF* - remove the ECF instance because it has become obsolete.

Implementation

The system is implemented on Sun workstations running Unix with TCP/IP and the window system SunView and written in Objective-C. The migration specifications are contained as objects which accompany their ECF (this is possible because Objective-C allows objects to be "passivated," stored, read in and activated again).

The overall migration system (MS) consists of a local migration server(LMS) running on each workstation and a global migration server (GMS).The LMS implements the office worker's interface to the MS, maintains the office worker's desktop model, and interprets the migration specification in order to find out which step will be the next after forwarding the ECF. The GMS controls the migration of ECF between stations. It also contains information for determining which office workers are logged in, the roles they are currently playing to resolve roles into concrete receivers, the ECF's state of migration, and events which trigger forwarding of ECF. Note that ECFs are copied to each workstation as it follows its migration course.

An organizational description is maintained as relations in the database system. The relations describe the valid roles and relations between roles and between organizational units (i.e., departments). This allows ECF templates to be defined which have a pre-defined migration path and the contents needed to perform the workflow procedure.

One key point that the system makes is that the description of the organization is set apart from the migration system algorithms. Also, the system designer realize that a single GMS will not handle large organizations. A cluster concept is being developed which will allow the integration of wide area networks.

2.1.3 Cokes

The COKES (Carlton Office Knowledge Engineering System) system, by R. Kaye and G. Karam [30], is an artificial intelligence approach to solving office automation. While it initially does not appear to be related, many of the same problems of routing and decisions as to where the knowledge is stored are found in this knowledge-base approach. The designers have chosen to distribute the office knowledge needed to solve a particular procedure to those users' sites which are responsible for carrying-out the procedure. Notions of personal agents, message forwarding and conditional suspending are present in this model

The system provides high level support of office workers by embedding office knowledge in a network of distributed cooperating knowledge-based "assistants" and servers. These assistants and servers incorporate both factual and procedural knowledge in terms of frames and rules, and are capable of making use of existing conventional office technology. The assistants are devoted to individual users, whereas the servers are dedicated to specialized areas of organizational knowledge or functionality. The system is capable of supporting concurrent multiple tasks with facilities for interruption and resumption of tasks. The various assistants and servers cooperate in solving problems by passing messages between themselves. One of the basic principles of the system is that office workers should be assisted by an entity which is knowledgeable about the organizational structure, the tasks the worker is currently working on, and user preferences and personal procedures.

The system aids the user in learning about the organizational structure and procedures by querying the knowledge-base. COKES uses rules to define procedures and frames to define office structure.

There are two types of frames: class frames and instance frames. In this sense, it is very similar to an object-oriented approach. The interesting twist to the frame's instance variables (a.k.a., slots) is an additional qualifier known as a facets. The facets identify procedures or daemons that are invoked if certain operations are performed on a slot. For example, if we attach a daemon to an *if_added* facet of a given slot, then when a value is added to the slot, this daemon is invoked automatically. Other facets include: *if_removed*, *if_replaced*, *if_needed*, and *if_appended*. This feature is useful in maintaining consistency across a set of slots or even across frames.

The architecture supplies each user with a knowledge-assistant. Servers also exist to supply general office knowledge and a means of maintaining centralized control over changes to the knowledge. All assistants and servers are equipped with both inference engines and facilities for communicating and cooperating with each other. Another basic COKES principle dictates that information should be stored where control over its evolution is located. Therefore, information about a particular project is stored in the project manager's personal assistant.

Rules are applied toward the achievement of a goal using a backward chaining inference strategy. Distributed and concurrent office activities may be suspended until some waiting condition can be satisfied.

2.1.4 PAGES

The research system known as PAGES by Heikki Hammainen[23, 24] makes significant contributions to the area of office workflow, agents, and form automation. This approach also centers around agents. Forms are defined as a means of communicating between each user's agent. The agents have access to local formbases which serve as a decentralized approach to workflow and routing evaluation.

Agents act both as consumers and providers of services. The services are accessed by interchanging and sharing forms. The agent acts as an object manager processing, presenting, and storing forms that contain data, layout, and rules.

A form is an instance of an object-oriented class. An agent is assigned to each user or organizational unit (e.g., accounting department). Each agent has access to a formbase (a set of instantiated forms). Forms get mailed to and from other agents. Forms can be simple, such as a confirmation form. All functionality of the system appears in rules of the form classes and all data in instances of these form classes. Rules are described by a form-oriented rule language, FPL (Form Programming Language).

Workflow designers are expected to first define a state transition diagram of the workflow process they wish to automate. Once all the states are determined, forms are defined to provide the communication and informational needs of the collaboration between the states (simple forms are confirmation, reject, accept, etc.). Each state change corresponds with a form interchange transaction. A Form Interchange Protocol (FIP) makes sure that when an agent transfers a form to a new agent/formbase, the form definition is equivalent or at least compatible. Two FIP-aware classes exist to define and support the form migration scheme: FIP message and FIP monitor classes. The instances of the FIP message classes implement the steps of FIPs, whereas the instances of the FIP monitor classes maintain the states of FIPs, but typically do not move themselves.

Note that agents always receive a copy of the form. If a receiving agent has no knowledge of the form class, the form definition will migrate and be installed into the user's formbase.

Forms are derived from an abstract base class, Form. The system supports the construction of applications within the local formbase. The fields of the form are drafted interactively with the form editor. Then the desired functionality is defined by filling in the standard rule slots of the class. Linking forms aid in constructing semantic networks. For example, when a delivery form completes, the action is linked to an inventory form which reduces the inventory by the ordered amount. Linking to another form occurs by specifying the class name, formID, and field name.

Agents are interconnected with a network which supports unique agent addresses and reliable transfer of forms. The agent identifies the arrival form and places it into the right folder, which verifies the conformance of the instance with the local class definition. The migration of form classes is a way to distribute functionality among agents but carries the risk of accepting incomplete, unsuitable, or even hostile classes.

Implemented as policy instead of mechanism, the agent should support the user in remembering (1) when and what kind of form he is expecting from others, (2) what others are expecting from him, and (3) how his forms are related with each other through pending tasks.

The term "project" defines a unit of collaboration which involves a set of agents and a set of forms. The generic project aspects are captured in the base class Project which allows the user to browse his projects as a uniform to-do list. The basic project information consists of the initiator, participants, date of start, and finish, and an entry for each form involved. The state of a project is either pending, idle, or finished. A pending project requires the user's attention since there is a deadline. Each participant maintains the local state of a project asynchronously according to the form flow visible to him.

An entire workflow process can consist of a set of forms and projects. In Figure 2, two organizational units (the Customer and the Supplier) each have an agent assigned to them and a collection of forms (Tender, Order, Confirm, Change, and Status). An example of a project may be a new status request from the supplier to the customer.

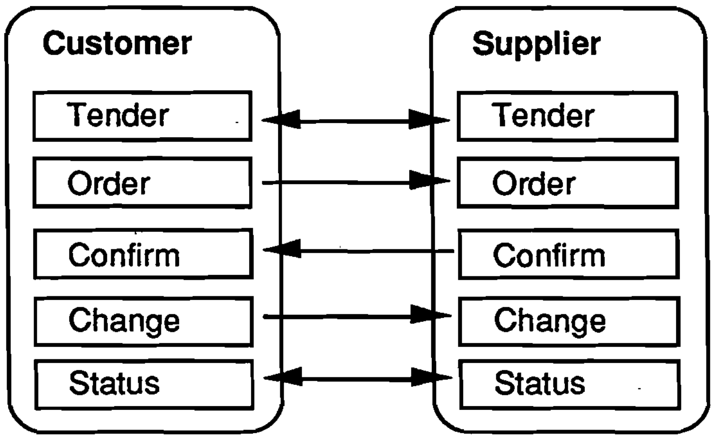


Figure 2: Forms interchanged during the ordering procedure.

To capture the past, a uniform bookkeeping of significant events is implemented with a form class History whose instances record standard events for example, form deleted, created, sent, or received, as well as any user-defined incidents. The main attributes of the History class is a time-stamp, reference to the associated form, and event type.

Real-time sharing of forms is achieved through a distributed windowing system, such as X; a single form can be viewed by multiple users in parallel.

The PAGES model does not impose a consistent global view to the organizational roles of agents. Instead, each agent maintains its own view of the world and implicit dynamic roles appear as side-effects of form interchange, that is, the sender of an order form becomes a customer and the receiver becomes a supplier. The rigid aspect of roles may appear locally in the rules of an individual agent. For instance, the team agent recognizes the team members and allows them to access the shared team forms.

A decentralized approach was chosen for the PAGES system. A single agent does not coordinate the entire workflow process. Instead, any participant which comes in contact with the workflow, can operate dynamically as the coordinator. Each self-contained agent is expected to know only its direct acquaintances (i.e., the agents it deals with).

Finally, note that the use of a form is slightly different than the traditional use. The PAGES model emphasizes the use of a form to aid in the communication between two collaborators or two users in a workflow; the form is usually designed to have only the needed information to complete a communication transaction. This approach differs from having a single form be used by all the participants in a workflow, customizing the presentation and filling-out rules based on the current user.

2.2 Commercial Systems

2.2.1 Electronic Mail

Various electronic mail packages can be considered a very primitive, unstructured workflow tool. Here, the messages and the text within the messages are not structured and each user decides in an ad-hoc manner the next recipients of the message. UNIX mail, CC:Mail[7] and MicrosoftMail[40] all support message transports that span heterogeneous machine architectures. This property must not be forgotten when designing a workflow system. Researchers contend that one of the major reasons why electronic mail is so successful is its ubiquitous access and availability on most machines.

2.2.2 BeyondMail

The BeyondMail product[5] from Beyond Inc. provides significant functionality over standard electronic mail packages. It can be considered a decentralized workflow system in which mail messages arrive for users in their inbox and rules are applied to them to file, perform some action or forward the message to a set of users. The system can be significantly improved if it has the ability to attach rules to the mail messages which would then be installed and executed at each site. This would allow for a much simpler distribution and maintenance of the routing script and state information.

BeyondMail allows users to create rules to manage their mailbox. These rules direct BeyondMail to automatically forward messages to colleagues, classify messages in personal folders, delete "junk mail", or generate automatic responses to routine messages. BeyondMail comes with 6 standard "forms" that can be sent to users. For example, there are "Phone Message" and generic "Request Forms" available in the set of standard forms. Each user can define a set of rules to be applied to the incoming mail messages and forms. The rules are in a "When...If...Then" format. A typical rule might be "When new mail enters my inbox, if the sender is my Supervisor and the word 'deadline' is used, then put the message in my Urgent folder." Users can switch between rule sets depending on the current situation (e.g., out of town, crisis mode). Rules can be triggered when certain events occur:

- New messages arrive in a user's inbox,
- A user reads a message for the first time,
- A messages that is filled into a folder,
- When BeyondMail is starting-up or exiting,
- Periodically (a recurrent time every hour, day, week, month)
- Timer (a specified date, time has arrived)

If a rule is sensitive to one of the events, it becomes active and the IF portion of the rule is examined to determine whether or not to fire the rule. Firing the rule will execute the THEN portion of the rule. Many different types of action can be performed during this stage:

- Construct a new mail message
- Send the mail message
- File the current message into a folder
- Display an alert box
- Perform some calculations (examine or set fields of the message)
- Launch an application with pre-defined keystroke inputs.

Constructing the rules are simplified by using rule templates and selection lists. A selection list displays to the user all of the possible settings. Many functions use selection lists as parameters. For example, the move command which places a message into a folder has a selection list which contains all of the available folders. The templates provide extra structure information when defining a rule. This reduces the amount of keywords a user must understand, prevents type mismatch, and quickens the amount of time needed to fill out the rule.

The rule language is modeled after HyperTalk in that the syntax is very english-like. Theoretically, anything a user can do manually, s/he can do it automatically using a rule and specifying the action as a verb within the rule definition. Rules can be generated by using the structured rule editor or by entering text adhering to the rule language syntax. Rule specification is simplified if a user wants the rule to be applied to a particular form since all forms have a rule template.

Composing a form message is easy if a user is using one of the standard forms; just fill in the fields and send it. Currently, there is no way of defining a new class of forms. Customization of the behavior of forms is achieved through the rule language. Documents or data files can be attached to messages. This means that the recipients can view the attached documents in their native application assuming that they know which application to use.

Because Beyond can make use of the structure of the mail message, it can more effectively manage a user's set of messages. The ability to perform full text queries on the main body of the message adds significant power to the management mechanism by being able to base its actions on the content of the message.

Routing is decentralized since the rules apply to a user's mailbox. It is possible to distribute rules to a set of users but this quickly becomes cumbersome (e.g., updating rules). This scheme tends to make it easy to program routing transactions that take one hop as opposed to a sequence of hops.

2.2.3 LotusNotes

A very popularly known system, LotusNotes [38], by Lotus concentrates on office workers communicating with one another and accessing public information. This system addresses some of the same functionality of form-centered automated workflow. It provides the means to hot-link data fields, access and manage collections of documents, track group activities, and has access control to the documents. The system serves more as a bulletin board and does not necessarily require documents to be routed.

Lotus Notes allows its users to create and access document-oriented information on PC LANs; among its features are a document database, store-and-forward electronic mail and conferencing functions. The package contains a standard set of applications for conferencing, project tracking, bulletin boards, contact management and status reporting as well as application development environments for customizing its built-in templates. The speed and flexibility of the tool allows frequent prototyping and customization for the particular user group which may be the key to the eventual success of the product.

The product's goal is to transform the way people work, particularly as it relates to how people deal with group or corporate information. Lotus believes that people interact in two ways: sending information back and forth to each other (e.g., using electronic mail), and accessing shared information, whether it be structured information in databases or unstructured information such as bulletin board conferencing. Towards this end, the application development environment can support applications that require dynamic addition of information, sharing across groups or locations, and access to text-based and graphics-based documents. For example, using Notes, a much more structured e-mail application could be built which does more filtering and categorizing of incoming information. This functionality, however, is only a portion of what other e-mail management products, such as BeyondMail, provide. Yet, the fact that such an application can easily be built is noteworthy.

Notes has been designed as a client/server application (one of the first in the PC world). The server functions as a repository and management point for the document databases (i.e., applications). In addition, the server controls the communication functions such as database replication and mail routing.

The notes applications are basically a database of documents where each document contains both structured and unstructured fields. The users and developers never see the actual database, except for an iconic representation on their desktop. All interactions with the database is through the use of Forms and Views. The forms are used to enter information into the database and present the information to the reader. Fields within the form can be many types: data fields, numeric fields, calculation fields, and fully-formatted, "rich-text" fields. Developers can restrict what type of information goes into a give field as well as provide a list of valid entries to choose from. Views are used to access the information. A main view sorts the documents (i.e., forms) by the most appropriate field (e.g., date, author). Secondary views may sort by different criteria. Views are created by application developers and new views can be added. Personal views can also be defined in which a user's private information in addition to the shared information can be presented.

A great deal of other functionality is provided by the Notes package. Documents can be hot-linked together even if the documents exist in different databases. A full-text engine adds additional searching capability. Filters exist to provide extra sorting abilities for documents and electronic mail messages. In addition, filters can be used to collect documents and perform a uniform action on all of the resulting documents such as add a new field or change the values within a field. A security scheme has also been set up to restrict access to databases as well as authentify users. Finally, the overall strength of the groupware application allows its applications to capture and make persistent a group's knowledge, expertise, and history.

2.2.4 Active Documents

Interleaf Inc. has defined a commercial system known as Active Documents[18]. Here the world centers around the document and attaching knowledge and behaviors to the documents to react intelligently in their environment. Because the architecture is so extensible, it is difficult to critique it since one could always argue that a program could be attached to the document to get the desired functionality. Nevertheless, the system lends itself to some interesting architecture designs. In terms of workflow, the system does not take on a centralized or decentralized philosophy but instead the workflow definition would be embeded in a form as an active document.

The system advances the current notion of document processing by defining an extensible, object-oriented system for describing and executing active documents. A run-time bindable object system and Lisp interpreter serve as the key underpinnings in supporting such a mechanism. Much of Interleaf's philosophy can be traced back to Emacs, one of the first widely-used extensible editors that allows its users the ability to mold the underlying editor for tasks sometimes not imagined by the software developers.

Interleaf describes active documents as structured documents and their processors in which the objects in the documents can be acted upon by, and can themselves act upon, other objects in the document or the outside world. A document is simply a collection of document objects (doc-objs).

The software is based upon an object-oriented system which allows standard class creation and inheritance mechanisms. The Lisp programs have access to a document's set of objects. These objects range from characters and graphics to higher-level objects that describe the structure of a document. Various events (e.g., opening, printing, or displaying a document or selecting objects) cause the objects to be sent messages allowing them to, for example, query external databases or remote processes to get the current state of external data.

Not only can the content be encapsulated as doc-objs, but the user interface and document behavior are described in terms of objects and therefore can be customized. Lisp permits the run-time reconfiguration of the user interface. For example, the keyboard mapping and the menu system can be dynamically redefined.

Associating methods with the documents is a very powerful concept. Permitting the methods to be defined at document execution time allows the active documents to be used in ways that the system architects may not have anticipated yet. For example, a method associated with a mathematical integral sign may originally only allow for the expression to be graphically displayed. That method could be replaced at run-time with a method that ships the expression to a symbolic mathematics processor for symbolic integration.

In fact, the documents can even communicate with external processes by using standard TCP/IP sockets. The Lisp load primitive permits an arbitrary ASCII file containing Lisp code to be read and interpreted.

In summary, Interleaf's active documents supports a document-centric approach to defining and customizing a document's appearance and behavior based on a run-time, object-oriented system. Documents can take on new functionality and responsibility in terms of the data that they manage and have access to.

2.2.5 Reach

A system currently under development by Reach Software Corp [35] promises to be a complete solution to workflow automation for PCs. It is a vision because the system is still being designed and a lot has been promised but we are still waiting until the product comes to market.

The Reach system will be a task flow automation approach to workflow automation. It will primarily consist of intelligent forms capable of routing themselves and changing states based on operations the user can perform at any given stage in the workflow. These intelligent forms will be attached to electronic mail messages, which will carry them around the network in a route defined by the designer. Two principle components will exist:

- WorkMAN/Forms. client based, scripting engine capable of processing workflows, running in the background, communicating with Mail system and DDE compliant.
- WorkMAN/Server. client-server based, using SQL to enable more complex querying of the workflow system, reports and administration.

Forms will be objects and consequently have a type (e.g., purchase order or an invoice) and a set of operations associated with it.

Dynamic routing will exist to determine which user on the system should get the form next. Route a form based on the state of the workflow and the specific operations that must occur next.

Role-based and rule-based constructs will aid the form routing. Each stage in the workflow will have a set of rules associated with it. Role-based addressing allows designers to define the route in generic terms rather than specific hard-coded users. For example, determine the manager of the user making the request as to who should receive the form next.

Scripting will allow applications to be launched and files specified to be processed by the application.

The form design tool will allow developers to define a form, the content of the form, the roles involved in processing that form, the applications needed to process the form and its route. Constructing the form will involve drawing text boxes, labels, and fields. Users can also make use of buttons and attach actions which will be executed when the button is hit. Dialog boxes will be used to specify standard operations with buttons. A script will automatically be generated that goes along with the button (e.g., launch an application). Routing rule templates will be used to provide common patterns for a variety of routing types: sequential, serial with loopbacks, branch routings with conditional operations. A graphical display of the route will be generated once the routing rules are entered. A workflow route can temporarily be altered if, for example, a manager is on vacation.

Hitting a Done button, for example, will execute the workflow script which specifies the completion of the current stage and requesting a new route. The workflow system will use MailMan primitives to create a message, address it to "manager," and attach it to a form. A user will send it off like a regular mail message. MailMan will resolve which user matches the specified role and send the message. When the next user receives the form, the software will check its type against the types in the form library and then check the state of the workflow, determining which operations must be enabled, and display the form on the screen accordingly.

Tracking the workflow at each step is proposed in the Reach system. A message will be sent to servers designated as "tracking points." This builds an audit trail of the workflow which will be saved in a database for later querying.

The Reach approach centers the entire workflow process around the use of a single form. A workflow engine background process will be running at each workstation. Processing of the form will be based on the form type and the state of the form. A centralized workflow description approach has been chosen (i.e., the entire workflow process definition is attached to a singular form). Currently, it will not be able to handle a set of forms.

2.2.6 System A (BLOC)

This system[11], as well as the following two systems, are still under non-disclosure agreements and, consequently, cannot be properly identified. However, System A has the ability to intelligently display and fill-in electronic forms across heterogeneous machines. The forms are active and dynamic objects that respond to the environment and user actions. Currently the system is being extended to take on more workflow functionality. Its main approach is to have one routing script be attached to a form which travels with the form. However, this company is trying to deal with the notion of a form set and data sharing between forms. It has denounced the idea of a centralized server being responsible for evaluating routing logic since its major strength is its target toward enterprise workflow which does not guarantee the accessibility of server machines.

2.2.7 System B (KeyFile)

System B [31] provides document management in the form of storage, retrieval and distribution. A variety of information types can be imported into the electronic environment such as scanned images, facsimile messages, DOS files, word processor files, voice messages, and handwritten notes. The system has three basic Document management functions: (1) Entering documents, an image-based version of a paper document; (2) Reviewing and Processing documents, sending and annotating documents and (3) Filing and Retrieving documents. The major strength of the system is its very rich tool integration which allows easy management of electronic documents.

The company defines workflow as processing and receiving documents or forms that follow this pattern:

- Create or receive a document
- Classify for appropriate handling
- File, throw out, enter into workflow "system"
- Perhaps copy it
- Distribute for action
- Take appropriate action
- Perhaps revise it

- Follow-up
- File or redistribute it

Some interesting points about the system is the use of an object-oriented document server. Documents are referenced and routed to users by unique documentIDs. Users at a given site have access to the document server and the representation of the document is an icon that references a document object. Multiple copies of the icon can exist since they all point to the same object. Commands exist to force a manual copy of a document. Having multiple icons that point to the same device is also useful if each icon is used within a different job or task. Different parameters could be set on each icon that matches the job. For example, a scanner could have different settings based on the document being imported into the environment.

Users can assign Properties to every item in the system to uniquely identify it. Some of the properties include: a Title, Type, Person, Description, and Date. Explicit keywording is also possible. An elaborate document search ability is available with multiple search criteria such as title, author, keywords, and date.

Scripts can be generated to automate a repetitive task. The way in which a script is defined is significant. Users use the drag-and-drop mechanism with icons. By properly dropping a sequence of icons onto a desktop job icon, the user defines the set of actions s/he wants performed. For example, dropping a scanner icon, an OCR icon and finally a file draw icon will import a paper document into electronic form and file it away.

Forms can be routed to any system user. The form authors can place a deadline on how quickly the user needs to process the form. A user can be notified when time is almost up. The Connection Manager provides a very intelligent transport mechanism. For example, it supports multiple hardware architectures, properly orders and aligns bytes.

To construct a routing form, there are 4 major components: send to someone FYI, send to someone waiting for a reply, suspend, and stop. The routing order is specified by a sequence of the 4 components and stored in a folder. Fields and buttons can be placed on the form. Action can be associated to buttons. For example, if the user hits the "Accept" button then forward the form to my supervisor; if the user hits "Reject" then return the form to the initiator. The suspend component adds extra functionality, or intelligence, into the workflow process as it can require that certain conditions be met before progressing. For example, suspend the workflow until all necessary documentation arrives.

The system adheres to Microsoft's Dynamic Data Exchange (DDE) protocol. This allows, for example, data from an application to automatically be imported into a workflow form, filling in the proper fields. Finally, data files can be intelligently attached to mail messages. That is, the message contains information such as what application to launch to view the data.

2.2.8 System C (Verimation)

In this system, we will only examine the routing primitives it has defined in its form language. A form's routing transaction can be one of two types: forward or destination. Forward destinations receive a copy of the form with all of the logic still active (e.g., verify fields) while final destinations only receive a static copy of the form with the data entered in by users. The syntax for specifying a route involves only a few simple calls:

```
)DESC
DEST       (userID1)
USRFWD     (No)
USRDEST    (No)
```

This information is declared in the description portion of the form. The above code indicates that the form should be routed to userID1 who will receive a final destination. The USRFWD and USRDEST codes indicate that users cannot dynamically alter the forward or final destinations of the form. Note that multiple users can be specified in the DEST command. The next example specifies that the form should be forwarded to userID1, and next to userID2, then two destination copies should be sent to userID3 and userID4. As soon as userID1 completes all of the desired input and hits the send button, userID2 will receive the form.

```
)DESC
    FORWARD   (userID1)
    FORWARD   (userID2)
    DEST      (userID3, userID4)
    USRFWD    (No)
    USRDEST   (No)
```

Variables may be defined and used as parameters to the FORWARD and DEST commands. This allows the form's route to be dynamically specified as conditions arise. Users can:

- add additional forward destinations,
- add, remove, or restore branches and
- replace, and re-order the destinations.

There are 21 reserved variables defined to support 21 routing hops (FWD.IDn where n = 0...20). Another reserved variable, &ZFWD identifies a count of how many hops have occurred. To add a hop, the form designer needs to assign a userID to the proper FWD.IDn variable. For example, to add user gf to the hop 5:

```
CNTL (.FWD.ID5 = DG.gf)
```

There can be no breaks in a route. If a forward destination is inserted, and it is not the next forward destination, the Form will automatically skip over that destination and go directly to the final destination. But if you need to add a hop in the middle of the route definition, how do you shift all the remaining hops down by one? This is done automatically. Removing a hop at a specified level is accomplished by assigning a blank to the control variable:

```
CNTL (.FWD.ID5 = ' ')
```

To add a branch to a previously defined route at level n, just another assignment to the FWD.IDn variable is necessary. To replace a forward destination, permission needs to be granted by the form designer: FWDCNTL(REPL). Issuing the assignment:

```
CNTL (.FWD.ID5 = &userName)
```

will replace all of the old destinations at level 5 to the new destination. Note that the only forward destination that the user can replace is the next forward destination.

Looping is difficult in this scheme. It is, however, possible to send the form back to the first user by resetting the .FWD variable:

```
CNTL (.FWD = 0)
```

The system always "remembers" the Memo-ID of the first user.

Note that only one set of forwarding controls can be assigned to a form. This seems like an unnecessary restriction. The form designer should be able to specify at each hop in the route whether the user has the permission to modify the route. The routing script is attached to the form instance and is evaluated locally at each user's site. This model assumes that forms will work in isolation.

2.2.9 Others

Form automation is a growing industry [46] and many more systems are bound to become available. Two systems, Rhapsody from AT&T, and NCR's Cooperation, are additional tools for providing workflow functionality for the office. Early workflow systems [2,25, 48,50,51] laid the groundwork for todays systems. Survey articles on office automation [6,16,27,36] provide a historical perspective as well as comprehensive studies on office needs. Information retrieval tools such as full-text retrieval systems[9] must be integrated into the automated workflow systems.

2.3 Summary

After reviewing the background literature and commercial systems, it is interesting to note that one can identify a common set of properties that most of the workflow systems provide. At the very least, forms or messages can be transferred to a recipient by specifying a role instead of a hard-coded userID. Most have chosen a decentralized, local evaluation scheme for calculating the next routing destination. Consequently, the systems tend to be limited in providing routing instructions that affect only one form and not a form set. Inter-form communication is very week and basing the next routing destination on state information contained in external forms is difficult. Almost all of the systems have provisions for conditional, cyclic and ad-hoc routing. Only the more sophisticated systems can handle parallel routing and merging. The remaining workflow properties such as resource balancing, atomic workflow transactions, and tracking have not been fully addressed.

2.4 Agent Systems

The term "Agent" has been used in a variety of ways within the computer industry. Here is a comprehensive list of some of the more popular systems and references to agents [1,3,4, 8,10,12,13,14,17,19,20,26,28,33,34,42,43,44,47].

2.5 Groupware

Any new software that is being developed must understand the issues involved in groups of users working together. Here is a list of some survey articles that discuss groupware issues[15,21,22,32,37,41,49].

3 Modeling Workflow Using Agents

3.1 Requirements

While designing our model, we attempted to adhere to a set of requirements or goals that we believe are imperative in a state-of-the-art workflow system. First the model should be robust enough to support a rich set of workflow transactions. That is, we should be able to define atomic transactions that have the same functionality of databases (e.g., can be journaled, and can recover from a system failure by rolling back to a consistent state). It is impractical to supply this level of functionality in a prototype but we believe it is necessary in a commercial system that must compete with manual procedures. The model should also not collapse by having single points of failure. For example, relying on a single, central database to always remain on-line is not realistic. The processing of the workflow should be distributed to make best use of the resources on hand. Obtaining higher throughput and responsiveness (e.g., in filling out the form) is very important. Workflow designers should be able to quickly define simple and complex workflows without expending a great deal of effort. Finally, the model, in principle, should be scalable.

3.2 Why use Agents

There are a number of approaches that one can take in implementing a workflow system: (1) rule-based systems like BeyondMail, (2) hard-wired programs and, (3) dedicated form languages such as PAGES and Logical Routing.

We chose to make use of agents because we had access to an agent framework (Brown's Envoy system) and wanted to see if the framework could be easily and quickly extended to support a workflow model.

It is also quite natural to consider agents as a key technology in building a workflow system since they poses functionality that overlaps with workflow functionality. Agents provide functionality such as:

- notification mechanisms,
- schedulers and managers of tasks,
- track information,
- and often server to locate users and system resources.

The Envoy system provides all of this functionality and we eagerly wanted to understand the complexities involved in upgrading the system to support a complete workflow design.

3.3 Agents in Our Design

We believe that agents provide a natural way to characterize and implement workflow automation. An agent can be defined as a computer-based assistant that helps users perform tedious, repetitive or time-consuming tasks more easily and efficiently. It can be implemented using agents which take on responsibilities for automating a series of user or system actions — performing these actions when proper conditions are met without any user intervention. In short, agents serve as higher layers of abstraction and offer the property of indirect invocation.

In our model, we have three types of agents:

- *Envoys* – which serve as delegators and managers of task assignments and results. An Envoy keeps track of a set of tasks and can inform users when a task changes status or has results, using a variety of notification channels.
- *Operatives* – these agents are attached, typically, to end-user applications to perform actions or work on behalf of a user. Most operatives have a single backend server component and multiple frontend components that users interact with.
- *Bureau Chief* – provides a site-wide lookup service for contacting agents and system resources. It abstracts naming and authentication services.

Each type of agent plays an integral part in orchestrating and implementing our workflow automation system and will be discussed later in detail.

3.4 Model Components

Below we describe the primary components of the model and outline the major responsibilities of each component (see Figure 3).

Our workflow model can be classified as, primarily, a centralized workflow model. However, one central form database (formbase) does not exist in this model. Instead we propose an actual, small database for each form class because of the need for robust (i.e., transaction-based) processing. Each formbase has a governing process (FormOperative Backend) that services form-level transactions such as creating a new form instance and supplying field authorization for all instances of its form. Our model is decentralized in the sense that the FormOperative Backend invokes a process on the user's local machine to handle field-level transactions (i.e., filling in the form).

A form's Backend process is comprised of one form class and a set of complementary classes that are bound in while the Backend is compiled. The complementary classes support the integration and interaction of the form class into the workflow model (e.g., communication mechanisms and protocols). The FormOperative Backend process is also responsible for executing the routing logic for each form instance by accessing methods of the form class. The form class defines a unique chunk of routing code for each workflow schema the form can participate in. The logic code does not travel with the form instance nor does there exist routing logic at each user's machine[2]. One FormOperative Backend process services all instances of its form class. For example, consider the FormOperative Backend that services a Travel Expense Form. If five users are in the middle of a Travel Report workflow then five instances of the Travel Expense Form may be in use but only one Backend process will be needed to service all of the Travel Expense Form requests.

Users interact with a form instance by using a FormOperative Frontend that runs on their local workstation. This Frontend process deals with displaying the form instance to the user and allows him/her to interactively fill-in the fields. Many conditions can cause the form instance to be advanced to the next workflow stage. The user could explicitly hit a "submit" button or the form could automatically advance to the next station under conditions such as (1) an error condition occurred, (2) all the fields were filled-in, or (3) a complementary form instance was completed. Once the form closes the Frontend process terminates.

Users each have an agent, called a User Envoy, assigned to them to work on their behalf. The User Envoy serves as a delegator and manager of task assignments and results. It aids in keeping track of the user's set of tasks and informs the user when new information for a task arrives by using a variety of notification channels (e.g., e-mail, alert box). A User Envoy would typically have a variety of registered tasks, or missions, such as: receiving and filtering mail, managing the user's daily calendar, and handling workflow requests.

In addition to the User Envoys, there is a Workflow Envoy (WE) for every workflow schema. The Workflow Envoy primarily serves as a coordinator and tracker. Only one Workflow Envoy process is needed to support multiple workflow instances belonging to a given workflow schema. The Workflow Envoy serves as the communication hub for its

[2]We use the terms *travel* and *hop* to mean that only the minimal form data is transferred during a one step routing transaction; the form is not physically moved or copied. Most likely, only a unique form identifier (formID) will be specified. At most, variable data is moved, not the form class definition which is accessed and cached.

workflow schema. This allows the Workflow envoy to keep a workflow transaction log for each workflow instance. It also maintains global workflow state information such as the addresses of all the participating User Envoys and Operative Frontends and Backends.

A single Bureau Chief process provides site-wide lookup and role-resolution services. The Bureau Chief process runs on a server machine and can supply information on how to contact User and Workflow Envoys as well as Operatives. It also maintains the role-resolution database that maps generic terms (e.g., Department Head) to specific userIDs.

The model makes use of a form set definition (FSD) which allows workflow designers the ability to define a collection of logically related form classes to specify which forms should be presented to the user during each workflow stage and which fields or groups of fields among the form instances should be hot linked. The Workflow Envoy holds one form set definition which it examines to ensure that the necessary forms are present during each workflow stage and to propagate the hot-link data (see figure below).

Figure 3 shows the primary components in the workflow model and their communication channels. Note that each user has one User Envoy and at any given time can have one or more operative Frontends (one per form instance) running while processing a workflow request.

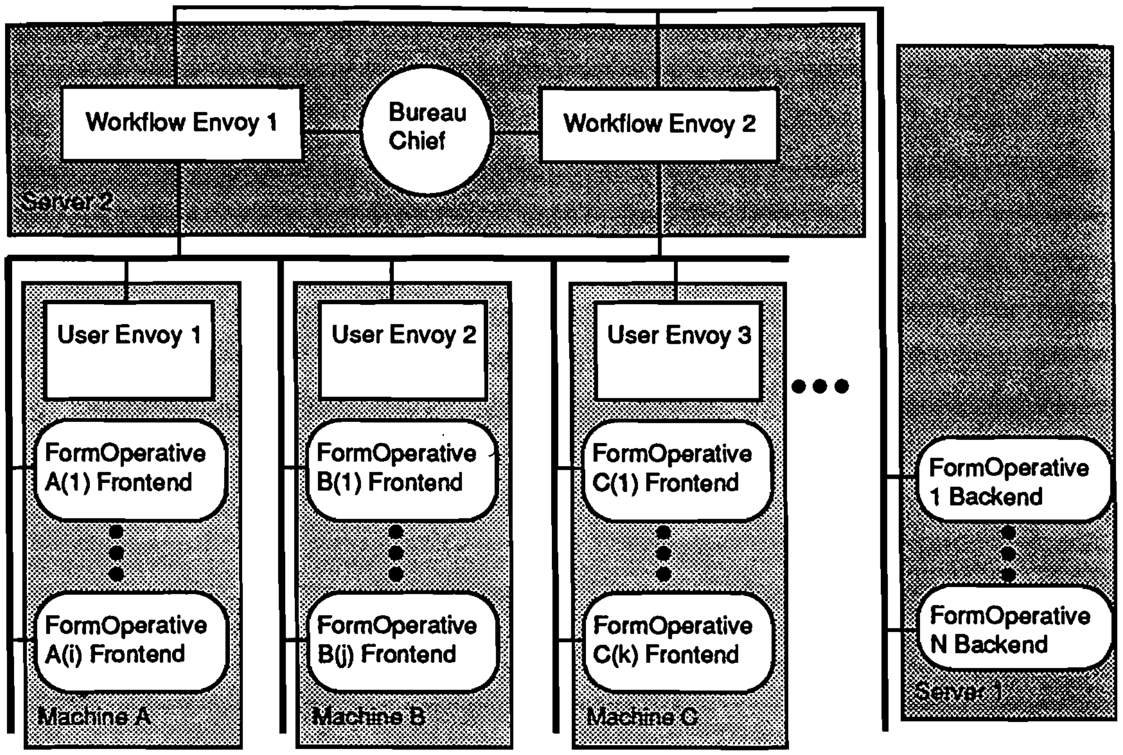


Figure 3: Workflow model components.

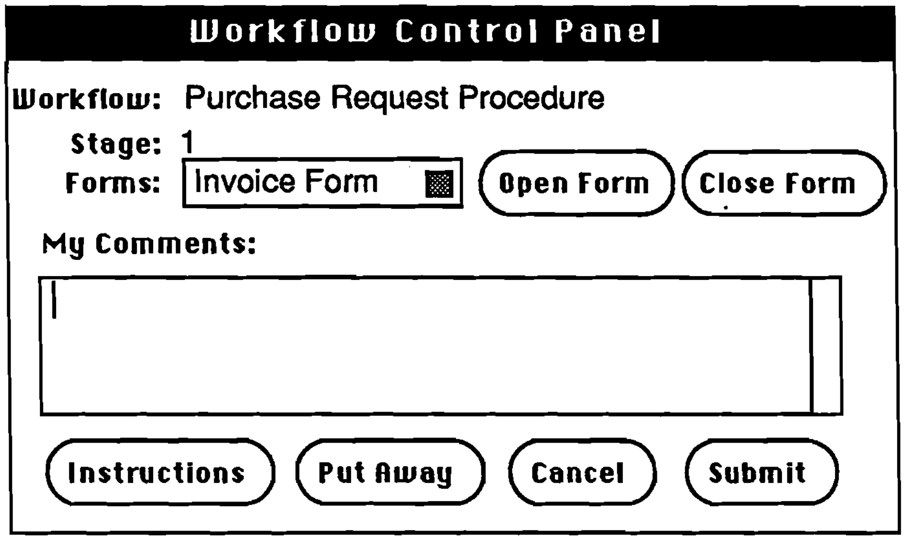
In summary, the model dictates that form-related functionality should be centralized by having individual FormOperative Backends servicing form-based transactions for a given form class. The routing logic for a form is defined within its form class. A separate chunk of routing code exists for each workflow schema the form belongs to. FormOperative Frontends service field-based transactions for a given form class. Workflow Envoys are a mechanism for centralizing the information needed to support office workflows that span multiple form instances. The form set definition (which also contains the field hot linking), tracking and global workflow state information all reside in the Workflow Envoys.

3.5 Initiating a Workflow Job

There are two types of workflow initiation: direct and indirect launching. A direct launching requires the user to issue a command, perhaps by selecting a menu system command (e.g., "Travel Expense Report") from his desktop. Alternatively, a manger may perform an indirect launching by issuing a command in which the first recipient is not the manager but one of his employees. In both cases, the initiating command will cause a message to be sent to the designated Workflow Envoy. If the workflow instance was launched directly, the Workflow Envoy will launch the appropriate Form operative Frontend(s) on the user's machine, supplying it with a pre-defined initial mission. The mission will also be installed into the User Envoy's set of registered missions. If the workflow was launched indirectly, the Workflow Envoy will only locate the first recipient's User Envoy and pass it the mission.

3.6 Workflow Control Panel

The workflow control panel is launched when a user opens a workflow mission from their User Envoy. The panel tells the user the workflow schema the mission originated from, the stage the workflow job is currently in, and the forms associated with the mission (pull down menu). Users can open and close forms that are associated to the mission, provide textual comments about the workflow job, see additional instructions, temporarily put away the set of forms for this mission, cancel the work done so far, or submit the work request. Note that the workflow control panel may obviate the need for a "Done" button on each form. The "Done" buttons may be confusing to users if they are presented with a set of forms to satisfy a work request.


Workflow Control Panel

Workflow: Purchase Request Procedure
Stage: 1
Forms: Invoice Form

Figure 4. Workflow Control Panel

3.7 Workflow Routing

The workflow routing of forms is achieved through routing logic code, state information, and envoy missions. Each topic is addressed below.

3.7.1 Routing code

The routing logic for a workflow is aggregated at the form level and each form operative Backend executes the routing logic for its form instance by accessing methods of the form class. Since a form may be used in multiple workflow definitions, the routing logic must be able to change depending on the workflow procedure being followed. Towards this end, the routing code is broken up into separate functions that get executed based on the incoming request. Each form operative Backend has a dispatcher routine which examines the incoming request and executes the proper routing procedure.

3.7.2 Routing state information

A routing table provides routing state and transition information for instantiated forms. When a new workflow job is encountered by the form operative Backend, a new form instance and routing table is generated. The table has 4 columns and a varying number or rows:

Role	Procedure	RequestCount	Status
Anyone	Initialize()	0	INITIAL
Manager	ManagerApproval()	0	INITIAL
AccountingDept	AccountLog()	0	INITIAL

The primary key for the table is the Role column. The incoming request has a role field which is compared with the state table. Once a match is found, the corresponding procedure is executed. The procedures can process the form (e.g., compute any additional values) and then forward the form to the next recipient. The RequestCount field indicates how many request messages have been received by the operative Backend for a given role. This information is important if the routing logic changes based on the number of trials a work request was attempted. For example, after the third time of trying to get an authorization, send the form to the department head. The status field conveys what state the work stage is at any given time. Some status entries are: INITIAL, REQUEST, REPEAT_REQUEST, WORKING, SUBMITTED, PROBLEM, and DONE.

The above table could be the initial routing table for a Purchase Form. When a purchase form is instantiated by a Workflow Envoy, the userID and role are initially supplied. The operative Backend's dispatch routine examines the routing state table and calls the associated procedure that matches the specified role (Anyone in this case). The Initialize() procedure is called which forwards a copy of the form to the userID by defining a mission.

3.7.3 Missions

Conceptually, a mission embodies a work request, typically for a user to process. A mission is simply a collection of data values consisting of information such as: userID, role, field authorization, work status (initially REQUEST), formID or bundleID, workflowID, and a timeStamp. The field authorization information indicates initially which fields should be visible and modifiable to the user. The mission is passed to the Workflow Envoy which determines the userID, if necessary by asking the Bureau Chief, and sends the mission to the user's User Envoy. When a user decides to process a work request, he opens the form(s) by selecting the mission from the Mission Summary application which is associated with his User Envoy. The User Envoy launches the needed form operatives Frontends to display the forms associated with the mission. In addition to the forms being displayed, a small workflow control panel opens.

When a mission is submitted by the user, the operative Frontend packetizes the mission data and contacts the Workflow Envoy. It, in turn, receives the data and sets tracking information for the mission such as userID, role, status=SUBMITTED, and a timeStamp. The information is relayed back to all of the operative Backends which participated in the work request. An acknowledgement message from all of the operative Backends is sent back to the Workflow Envoy.

If a mission submission is overdue, the Workflow Envoy is responsible for sending a message to the user's User Envoy which, in turn, notifies the user of the problem. When the Workflow Envoy originally received the mission, it recorded the current time and the mission overdue time. An alarm time was calculated and queued. A submission of a mission causes the alarm to be deactivated.

Backend Operatives Receiving Completed Missions
When a work request mission returns to the form operative Backend, the dispatch routine examines the userID and role and locates the proper entry in the routing state table. The RequestCount counter is incremented and the status changes to SUBMITTED. Next, the routing procedure is executed. If any data was added or changed on the form, it is written to a file on disk for persistent storage.

Mission Results & Reports
In Brown's original Envoy model, missions are carried-out by operatives and they generate three report types (message, short report, and interactive report) which provide progressively detailed amounts of result information. In the extended workflow model, the missions are carried-out by users. Consequently, missions do not need to return results but instead status information. The simplest form of status information may be to only get an acknowledgement message back from the operative Backends stating that they received the

user's submitted mission data. Providing more information may be trickier. Users probably want to receive status information as the operative Backends and Workflow Envoy initiates and completes future stages. This information will be stored and presented by the Workflow Envoys. It would be impractical to constantly propagate the status information to all the User Envoys that have participated in the workflow.

3.8 Example of Component Interaction in the Model

To understand how the components interact with one another, consider the following example. A user, Bob, wishes to get reimbursed for a trip he recently took and initiates a Travel Expense workflow by selecting a menu command from his desktop. The Workflow Envoy that has the Travel Expense workflow schema defined is sent a message to initiate a new instance of the workflow.

The Workflow Envoy next examines its form set definition and determine which forms need to be sent to Bob for the initial workflow stage. Initially, only a Travel Expense Form needs to be filled in by Bob so the Workflow Envoy contacts the associated FormOperative Backend process that handles this particular form and requests that a new form instance be created (the Bureau Chief is consulted to obtain the Backend's address). The Backend process initializes some form-specific state information and sends the Workflow Envoy a unique formID for the new form instance. The Workflow Envoy then asks the Bureau Chief for the address of Bob's User Envoy. Because the user who initiated the workflow is also receiving the first workflow stage, the Workflow Envoy can directly launch the Travel Expense FormOperative Frontend on Bob's machine. Otherwise, the Workflow Envoy would send a request to the user's User Envoy.

The Frontend displays the form for Bob and he fills in the necessary fields. When he hits the "Submit" button, the Frontend packetizes the field data and sends a message to the Workflow Envoy. The Frontend process terminates. The Workflow Envoy records the event in its workflow transaction log and relays the message to the Travel Expense FormOperative Backend.

After receiving the message, the Form Operative Backend executes the routing logic code for the next workflow stage which involves sending the form instance to Bob's Supervisor, Alan. The Workflow Envoy receives the message and queries the Bureau Chief to resolve the Supervisor role and obtains a specific userID. Once it receives the userID, the Workflow Envoy accesses the form set definition to determine if additional forms need to be sent along. In this case, no additional forms need to be sent and Alan's

User Envoy is contacted and passed the work request. The Workflow Envoy records the event in its workflow transaction log.

Alan's User Envoy informs him that a new workflow request has arrived. When Alan decides to process the request, the appropriate Travel Request FormOperative Frontend is launched by the User Envoy and the form is displayed. The Supervisor approves the form which causes the form to close and the Frontend process packetizes the data and sends it to the Workflow Envoy.

Once again the Workflow Envoy records the event in its workflow transaction log and relays the message to the FormOperative Backend. No additional user intervention is needed at this point. The Backend executes the routing code which causes a few calculations (e.g., sums up the total travel expenditures for the user) and the data is stored in a database for future reference.

If the Supervisor did not approve the travel report, the data would still be packetized and sent to the Workflow Envoy. The event would be recorded and the Workflow Envoy would contact the FormOperative Backend which would execute routing logic. This time, the Backend would indicate that the form be forwarded back to the initial user. The Workflow Envoy would receive this request and, already knowing the address of the initial user's User Envoy, send it the message. Bob will be asked to resubmit the Travel report.

This example shows when the components are invoked and how they interact with one another. A more complicated example involving multiple forms in a form set and hot linked data fields would emphasis the importance of the Workflow Envoy as a coordinator and tracker. Next, we discuss how form sets are defined in our model.

3.9 Form Sets and Bundles

Workflows that use more than one form require multiple threads of control for each form. The Workflow envoy is responsible for managing the threads of control and synchronizing them when a work order is requested. Form sets determine which forms should be present during a workflow stage and, consequently, automatically indicate to the Workflow envoy that it should suspend the next workflow stage until all of the required forms have arrived.

Form sets and routing bundles serve two separate functions. The form sets provide the support for hot linking fields between forms and specifying which forms should be presented to users during a workflow stage. The routing bundles are a convenient way of naming a collection of form *instances*. For example, a user may wish to performs a query and retrieve all the formIDs of a given form type that were created last month. This bundle of forms could be used as input to a workflow procedure.

Form sets are a collection of, presumably related, form *definitions*. The main purpose of defining a form set is to be able to link data fields within the forms in the form set so that when data changes in one form, the change automatically propagates to the other form instances in the form set. A form set also allows designers to specify which forms should be sent and presented to a user during a stage in the workflow. To support this type of functionality, a form set data structure is installed at the Workflow Envoy. The data structure first consists of entries for each form class within the form set containing the following pieces of information:

- FormOperativeClassName,
- the machine and port number the operative is running at,
- the field names that are hot linked,
- the formID.

The next portion of the data structure consists of routing information which specifies the workflow stage name and the collection of form instances that should be routed and displayed for the recipient. For example, a purchase request workflow consists of three forms and three hops, with the name, address, date and purchase ID fields hot linked. The first form instance should be displayed to the initiating user and to the second recipient, a manager, while all three forms should be presented to the third and final user, the accounting department. Below is how this form set data structure might look like:

```
workflowID=66532
1. PurchaseForm, ivy, 1134, {Name, street, city, date, purchaseID}, 10765
2. InvoiceForm, irwin, 2007, {Name, street, city, date, purchaseID}, 10766
3. ApprovalForm, iris, 233, {Name, street, city, date, purchaseID}, 10767
InitialStage: 1
ManagerStage: 1
ApprovalStage: 1, 2, 3
```

Note that it is necessary to be able to specify the hot link fields in each form entry so that designers have the flexibility to enable/disable the hot linking between form instances. The field names need to be unique across the form set.

To implement the hot linking of data fields, each operative will call a PropagateFields() function after the role-procedures (i.e., routing logic code) have completed and perhaps more frequently if necessary. This function determines which fields have changed and then examines the form set data structure to send the data directly to those form operative Backends which have hot links. This would require the Backends to contact the Workflow Envoy, however, a copy of the form set definition may reside with the Backends as an optimization. The form operative Backends that receive the update message will determine if the data needs to be immediately sent to the operative Frontends if users are actively working on the particular form instance and have requested immediate notification.

The notion of Form Sets allows designers the ability to decouple the form's definition from explicitly specifying hot linked data fields. It gives workflow schema designers some dynamic flexibility in determining which forms should be present at each workflow stage. A new Form Set can be installed at the Workflow Envoys and Operative Backends without recompiling and potentially without disruption in service.

3.10 Spawning Forms

If a workflow job requires that an additional form be used, the current form operative Backend will issue a spawn command that sends a message to the Workflow Envoy. The message will indicate whether the request is to retrieve an existing form instance or create a new one. The Workflow Envoy, in turn, locates the proper form operative Backend and passes it the message. A formID is returned to the Workflow Envoy which relays the id back to the original form operative Backend. Most likely, the Backend which issued the spawn command will either access data fields on the form or forward the form to a user. To forward the form, the operative designer needs to issue the FRforward() command. To access field data on the form, the designer makes use of two functions:

```
int  FRgetFieldData(formID, fieldName, INT | STRING | BYTES, &data);
```

This function retrieves the specified fieldName from the formID and places the values in the data variable. Note that the userID will be checked to make sure that the user has read access to the field.

```
int  FRsetFieldData(formID, fieldName, INT | STRING | BYTES, data);
```

This function attempts to store the data value into the fieldName field on the specified form. The userID will also be checked to make sure that the user has the proper write permission.

Workflow designers can specify in a form set the option presence of a form for a workflow stage by entering a negative index value. The workflow stage that contains a negative form index value will not be presented to the user initially. However, if the form is spawned by an operative Backend, the negative value is converted into a positive one throughout the form set definition. This causes the form to be dynamically added to the form set.

3.11 Splitting and Merging Forms or Sub-Forms

If the workflow schema can support parallel work requests, the form operative Backend will send a set of users a work request each having write access to mutually exclusive portions of the form. It is up to the form designer to decide if it is necessary to wait until all the work requests come back before moving to the next workflow stage or if it is possible to make decisions based on the results of some of the work requests.

Because the form operative Backend is a server and could receive many types of messages (e.g., status requests, new form instantiations, hot link data updates) it cannot block, waiting until work request missions return. Instead, form designers need to add this level of intelligence into the routing procedures. A convenience function needs to be provided to indicate when all of the pending work request missions have returned (see Form Routing Library).

Note that when a form is split, each of the receiving users can inherit an independent thread of control if the workflow dictates. Threads of routing control are identified by the operative Backend by the first user who receives the split form. Typically, a form will split and then be reconciled during the next workflow stage.

3.12 Conditional Merging

Conditional merging provides a finer granularity of control during the routing process by allowing the designer to perform actions before all of the forms to be merged have arrived. For example, consider the following scenario in which a workflow causes the form to be split after the initiating user (I) hits the done button. Users A and B receive the form in which they need to fill-in mutually exclusive portions of the form in parallel. The form travels to user C for approval or rejection (see Figure 5).

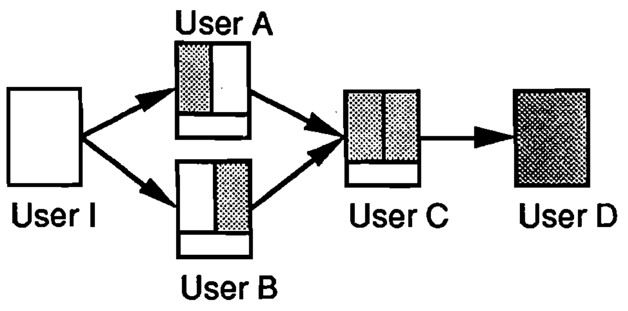


Figure 5: Conditional merging example.

With conditional merging, the form automatically advances to user C when either user finishes filling-out their portion. User C has the option of viewing the partially filled in form and taking action immediately. The routing logic could be designed to advance the form to user D if one copy of the form has arrived and been approved:

- user A's form has arrived and been approved, while user B's form still has not been received
- user B's form has arrived and been approved, while user A's form still has not been received

Alternatively, the logic may dictate that both forms be examined and approved before advancing to user D. Similarly, the form can be rejected if only one copy of the form is received, reviewed and rejected or both may need to be examined before a decision can be made.

3.13 Tracking and Mission Summary

Because all missions and communication messages are relayed through the Workflow Envoy, a rich set of tracking data can be recorded. A variety of applications can be built that communicate with the Workflow Envoy to access the tracking data. A Mission Summary application may list all of the registered workflow jobs that the Envoy has participated in and sort them by status information such as: active/completed, userID, percentage done, and priority.

3.14 Form Operatives

A new form operative base class (cEvFormOp) will be defined which communicates with a Workflow Envoy (WE) and User Envoys. The new operative class will be a *continuous* operative, running all of the time, servicing workflow requests. When a user initiates a workflow process, the appropriate Form operative Backend is contacted by the Workflow Envoy or instantiated if one does not initially exist.

Each new form added to the system will need to subclass the cEvFormOp to provide form-specific information such as the:

- content – the fields,
- presentation – layout of the fields,
- behavior – actions associated with the fields such as verification, and
- routing – the ordering of who should process the form.

Presently, we are not concerned with providing tools for form designers to specify the content, presentation or behavior of the form. Instead we have concentrated on supplying a set of routing primitives for designers to use to make defining workflows functionally rich and easy to program.

3.15 Role Procedures

The form routing logic has been partitioned into operative Backend functions known as role-procedures. The functions can be accessed by generic roles as opposed to specific user ideas. Moreover, the same functions can often be used in a variety of workflow schemas. When an operative Backend receives a submitted form, an operative dispatch routine determines who the sender was and accesses the current role-procedure for the user. This allows the workflow designers the ability to post-process the form's content and implement conditional routing logic. Once the current role procedure finishes executing, the next role-procedure is computed. It is subsequently called which primarily advances the form to the next step in the workflow.

To make programming role-procedures simpler, a set of routing primitives have been defined which deal with transporting forms to users, inter-form communication and data sharing, and synchronizing threads of routing control.

3.16 Form Routing Library

We have designed a minimal but complete routing library to be used within a high level programming language (e.g.., C++) or a 4GL oriented form language.
Below is a list of the functions described in the Form Routing Library (FRL) along with a discussion of how the function will be incorporated into the workflow model.

3.16.1 FRforward()

```
int  FRforward(formID | bundleID, wkflStageName, missionData, recipient1,...,recipientN);
```

This call should be placed as one of the last calls in a role-procedure. It indicates that a mission should be constructed and sent to the Workflow Envoy. This will cause the Workflow Envoy to immediately resolve who the recipients are if roles were specified and notify the User Envoys that a new work request has been established. This call does not block; the current user's interaction with the form will end when the role-procedure ends. An original is always sent to the recipient. The workflow stage name (wkflStageName) is sent along with the forwarding request. This information is used when accessing the form set definition to determine what additional form instances need to be present for the given workflow stage.

3.16.2 FRrecall()

```
int  FRrecall(formID | bundleID, wkflStageName,  recipient1,...,recipientN);
```

The routing facility allows designers the ability to recall a form after it has left a users out basket. Users should be able to re-open the form, make changes and hit the "Done" button once again. When the next recipient actively works on the form (i.e., opens the form), it is no longer possible for the previous user to recall the form; the previous state is committed once the next user begins working on the form.

3.16.3 FRbundle()

```
bundleID   FRbundle(formID1,...,formIDn)
```

Bundle the specified forms and assign a unique bundleID to them. Note that only form instances can be bundled. A bundle can be considered a package that is sent during a hop.

```
int  FRbundleAdd(bundleID, formID1,...,formIDn)
```

Add one or more forms to the given bundleID.

```
int  FRbundleRemove(bundleID, formID1,...,formIDn)
```

Remove one or more forms from the given bundleID.

```
int  FRbundleJoin(destBundleID, srcBundle1, ..., srcBundleN)
```

Join together a collection of bundles into a destination bundle.

3.16.4 FRtimer()

```
int  FRtimer(index, timeout);
```

The function will evaluate to FALSE if the timeout value has not been reached. This function can be used if workflow designer want to place deadlines on completion of workflow stages.

3.16.5 FRsuspendFormArrival()

```
int  FRsuspendFormArrival(formID | formSetIndex, roleProcID, status, timeout);
```

The function takes a specific formID or a form set index number to suspend execution until the form enters the specified roleProcID and has a matching status flag. The timeout value prevents the function from suspending indefinitely.

3.16.6 FRsuspendMerge()

```
int  FRsuspendMerge(formID, timeout, recipient1,...,recipientN);
```

This function can be used to enforce that all recipients have finished processing their portion of the form before forwarding to the next workflow stage. Because the recipients never have a copy of the form but are interacting with the form operative Backend (server), then merging and multiple threads is not a problem.

3.16.7 FRcopy

```
int  FRcopy();
```

Makes an explicit copy of the form instance for the current user. This causes a file to be written to disk containing all of the data currently stored in the form's fields.

3.16.8 FRspawn

```
formID  FRspawn(retrieveFormID | formClass, RETRIEVE | DISPLAY );
```

Spawning a new form will cause the associated form operative Backend to be invoked. Flow of control still remains with the calling (i.e., parent) operative Backend. Inter-form communication may be required if the parent operative Backend needs to wait until the sibling operative Backend has completed or needs to extract data from the sibling's form (see FRgetFieldData() and FRsetFieldData()). The last parameter tells the Workflow Envoy whether or not to display the retrieved form or not. One may want to create a new instance of a form class to be sent to the next recipient and, consequently, may not be interested in seeing the empty form displayed on the current user's workstation.

3.16.9 FRresolveRole

```
userID   FRresolveRole(roleName, LOCAL | GLOBAL);
```

To provide a means of specifying the workflow generically, roles may be used. Roles can identify an organizational position (e.g., department head), department (e.g., accounting), or relationship (my supervisor). Note that it is advisable that a Role resolution be computed at the last possible moment in order to provide the most flexibility and accuracy in the resolution algorithm. If the function is called using the GLOBAL parameter, a database lookup will be forced. Otherwise, if a LOCAL call is make, an attempt to resolve the role in its private Role table will first be tried. If no entries can be found, the database is consulted. One application of using the LOCAL parameter may be during a cyclic route when once a Role is associated to a user, the form needs to be routed to the same person throughout all of the looping cycles. The first encounter with the LOCAL resolution call will require a database lookup but subsequent calls will resolve the Role in the private Role table. Note that an additional component must exist which allows the designers to define their organizational structure and construct the role database.

3.16.10 FRinstallRoleProc

```
int  FRinstallRoleProc(role,procedure());
```

Once the designer has determined all of the roles and procedures needed to process the form for a given workflow schema, s/he needs to install this information using the FRinstallRoleProc() function. Note that the set of role-procedures needs to be installed every time the operative Backend is initialized. The role parameter should be a null-terminated character string and the procedure parameter needs to be a function pointer.

4 Implementation

Our prototype system, known as Workflow Envoys, has been implemented on Sun SparcStations running UNIX, using C++, X11/Motif, and TCP/IP for all network transactions. The Envoy Framework was also used as the core skeleton for supporting our workflow model.

One of the goals of implementing portions of the workflow model was to see how easily the Envoy framework could be extended to support workflow functionality. Towards this end, we tried to preserve as many of the working protocols that the Envoy system had already established. This guided many of the implementation decision in terms of division of labor between components.

Although we wanted to only concentrate on the routing primitives of the Form Routing Library, it was necessary to implement a great deal of the internal infrastructure to support the complete design. Practically all of the necessary communication protocols between the components needed to be implemented.

Only the core functions within the Form Routing Library were implemented to test the validity of our design. Specifically, the following calls were made available: FRforward(), FRspawn(), FRgetFieldData(), FRsetFieldData(), FRinstallRoleProc().

The entire implementation effort was compressed into slightly over three weeks (see Appendix IV: Implementation Schedule). Below is a more detailed discussion of the implementation along with the user interface and a sophisticated workflow example which was programmed in order to test and critique the model and implementation.

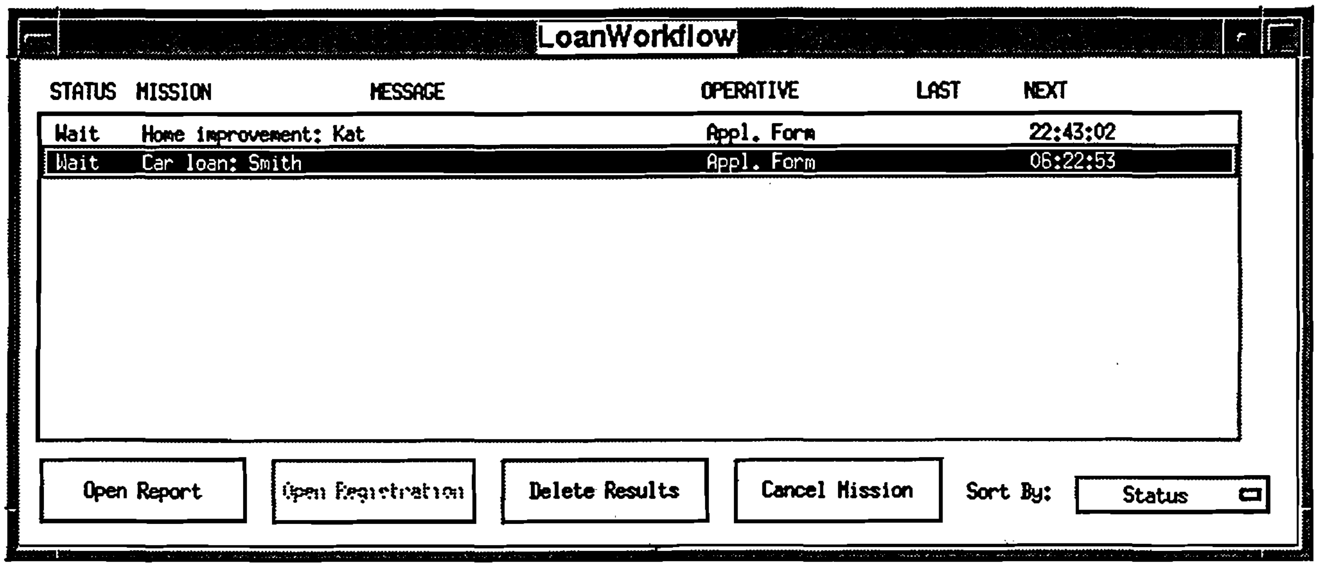
4.1 User Interface

In our model, users have one User Envoy assigned to them and a representation of this agent is embodied as a desktop icon. The icon has three basic states (see figure 6): (1) envoy active, (2) envoy waiting to launch a mission or waiting for user intervention, and (3) the envoy has received a completed mission report from an operative (e.g., a copy of the submitted form set).

Missions
Missions

Figure 6: Envoy icon states: (a) Envoy waiting to launch a mission or user intervention, (b) The envoy has received a completed mission report.

A user will most likely have a variety of missions registered with its User envoy. For example, a user may have a mail filter mission and a set of full text monitoring missions on a few public directories (see Figure 7).



STATUS	MISSION	MESSAGE	OPERATIVE	LAST	NEXT
Wait	Home improvement: Kat		Appl. Form		22:43:02
Wait	Car loan: Smith		Appl. Form		06:22:53

Figure 7: Mission Summary Application

When a user clicks on his or her Envoy icon, the Mission Summary application is opened to present the user with a tabular list of his or her missions along with status information. Each entry in the mission summary corresponds to either a registered mission that the user initiated or a workflow instance "work request" which could have been issued by almost any user. If it is a work request, the WAIT status will be displayed along with the workflow instance name, date received and Form operative responsible for generating the request.

To process a work request, a user will select the proper entry in the Mission Summary application and hit the "Open Report" button. The user will be presented with the associated form set. The proper form operative Frontends will be launched to display the forms and interact with the user. Note that if the user initiated the workflow instance and is the first recipient of the workflow, then the forms will automatically be presented to the user. S/he does not need to explicitly go through the Mission Summary application and manually open the workflow. At anytime, the user can opt to Delete mission entries as well as Cancel the mission. Cancelling a work request will cause a message to be sent back to the form operative Backend which sent the request. It is up to the operative Backend designers to determine what to do; one possibility is to re-route the request to another user.

Uses will also have access to one or more Workflow envoys on their desktop which they can open and see a list of workflow instances that have completed or are in progress. Each entry represents one workflow instance. Selecting a workflow instance and choosing the "Open Report" button will cause the Workflow envoy to present the latest submitted state of the workflow. While there is only one Workflow envoy per workflow schema, there can be multiple Workflow icons and Mission Summary applications that access the set of workflow instances.

A simple ASCII-based form presentation and fill-in Frontend was designed for users to interact with (see Figure 9). When a form is launched, a new widow (i.e., xterm) is opened and the form Frontend is executed and the form is initially displayed. Each field belonging to the form has an index number presented along with the field name, current values, and an access control indicator. There are three indicators:

- R Read access only
- W Read and write access
- F Required field which needs to be filled in before submission

Users have access to the following commands:

- 'd' Display the form and all of its fields.
- 'e num' Edit fields in the form starting at index *num*. The default *num* value is 0.
- 'f' Edit those fields which have been requested to be filled in by the operative Backend.
- 's' Submit the form
- 'q' Quit and revert the form back to its original state.

Surprisingly, this level of commands provides enough functionality to support simple form presentation and editing.

4.2 Components

When implementing the workflow components, every effort to preserve the compatibility with the existing Envoy model was taken and enforced. Below is a discussion of how each component faired during the implementation process:

- *User Envoy* - Very few changes were necessary. The only extension was the added concept of having a base missionID and complementary missionIDs. Here we needed the ability to uniquely have the User Envoy identify and generate the workflow instance responsible for the work request. When a user selects the mission in the Mission Summary application, the base missionID is consulted to retrieve the interactive report which allows the forms to be presented and filled in.

- *Workflow Envoy* - This component was based on the original User envoy and makes use of the same communication protocol between components. A simple workflow transaction log was implemented to preserve the workflow results at each stage. The Workflow envoy also caused the implementation of the Form set. It currently is defined by making simple function calls provided in the base Workflow Envoy class. Ultimately we would want the Workflow envoy to be able to read the form set definition in from a file. This would also all them to automatically handle different workflow schemas without being recompiled.

- *Mission Summary* - No changes were necessary in this component.

- *Bureau Chief* - This component was not implemented. As a consequence, the role resolution service was not available and userIDs were supplied whenever roles were needed. Note that this had little effect on the implementation of the components and libraries that require roles. In addition, environment variables were used to specify the current location of the operatives, User envoys, Mission Summaries, and Workflow envoys.

- *Form Operatives* – Most of the programming effort fell on this component and on the Workflow envoy. Some of the basic functionality was inherited from the original, server-based operative core class. A simple form Frontend was developed to allow users the ability to see and fill-in forms. Basic field-level access control was also established. In terms of the Backend, the entire role-procedure mechanism was implemented as part of the base operative class. Two sample operative Backends were constructed to support the following workflow example.

4.3 Workflow Example

Consider the following bank loan workflow schema. Two individuals wish to apply for a car loan at a local bank. Both decide to apply electronically and send a mail message to the bank requesting that they initiate the proper workflow instance. A Loan Officer receives the request and initiates the proper workflow schema. He must first fill in his name and the requested loan amount. Next, each applicant (the primary and secondary) are requested to fill in mutually exclusive portions of the form which they can do in parallel. If either applicant has assets greater than $100 dollars then an additional form is spawned for the user to fill-in. This form is an itemized assets listing and will be dynamically added to the form set of the workflow. Note that the workflow will be suspended until all pieces have arrived, including any spawned forms. Once both portions of the application are submitted, the form is reconciled and presented back to the loan officer. He calculates a risk factor and decides whether to accept or reject the loan application. If he accepts the loan and the requested amount is greater or equal to $10,000 dollars, then he must get his supervisor to approve the request. This conditional routing has been built into the workflow schema and his supervisor will automatically be sent the work request if need be. Once the supervisor receives the workflow instance with all of the accompanying forms, s/he can make a decision whether to authorize the loan. If the supervisor notices anything wrong with the application, s/he can send it back to the loan office. The loan officer can correct any mistakes or inconsistencies and resubmit it to his supervisor. This cycle can continue until a decision is made on the loan application.

This workflow schema was successful programed within our prototype. To demo the system we allowed all User envoys and the Workflow envoy to be accessible from one machine. Figure 8[3] shows the desktop with the initial four User envoys and the Workflow envoys. The next figure (Figure 9) shows the stage in which the form is split and both applicants have a work request. The final figure shows the Supervisor reviewing the form set which includes an Itemized assets worksheet form supplied by the primary applicant.

3 The next three figures are located in the appendix.

5 Conclusions

We believe that forms are intelligent, active, and dynamic objects that should interact with one another during an automated workflow procedure. Many of the existing research and commercial systems focus on forms working in isolation. This perspective is too narrow and promotes simple workflows that may not be sophisticated enough to replace the paper versions. In presenting our philosophy, we have outlined what we believe a state-of-the-art workflow system must have:

- form sets,
- atomic workflow transactions,
- user authentication and access control,
- customized viewing of forms and layout,
- hot linking between forms within a form set,
- the ability to split a form and merge it at a later workflow stage;
- conditional, linear, cyclic and parallel routing;
- the use of roles;
- varying forms of notification;
- tracking singular workflow instances and collective tracking, and
- resource balancing.

With this in mind, we designed an agent-based workflow model that provides centrally defined workflows which get executed on form serves who manage and route forms belonging to a given type.

To aid designers in developing workflow schemas, we established a Form Routing Library to be used with high level programming languages (e.g., C++) or dedicated form languages.

The core set of functions within the Form Routing Library were implemented along with the major components in our workflow model. This development was aided by the existence of an agent framework which was upgraded to support our requirements.

5.1 Analysis of Model

Our model assumes that users work in a computing environment that supports multiple processes and are networked together. We have tried to avoid being susceptible to single-points of failure by having one central database. Instead we have separate databases for each form type. These databases can be replicated and distributed to those sites that often use workflow schemas that involve the given form type.

Defining a Workflow envoy process that serves as an information clearinghouse serves many functions. Tracking information can be collected and applied towards adapting the workflow and tune it for peek throughput.

Although only parts of the model were prototyped, the short amount of time needed to build the system up to its current state leads us to believe that having an agent framework can reduce duplication of efforts. Now that we have promoted the agent framework into a workflow framework, it is interesting to wonder how this new framework can be extended again.

5.2 Future Work

One possible area for further investigation is to devise a means of visually representing and constructing a workflow route which can later be compiled. This verges on the discipline of visual programming and shares its difficulty when programs are large and complex. An initial iconic workflow language has been outlined in the Appendix titled "Visual Routing Language Scenarios". Note that the language is incomplete and serves as future research.

Still, there are plenty of areas that can be investigated within the current prototype. Tackling the other workflow elements (e.g., tracking, resource balancing, notification, etc.) and building upon the current design should continue to prove the viability of our approach to form-centered automated workflow.

6 Bibliography

Below is a list of references to workflow automation systems (W), agents (A), and groupware (G). Each entry will have one or more content indicators (W, A, G) which classifies the reference as dominantly discussing the marked topic. A bold entry symbolizes a pivotal reference that adds significant contribution to the topic area.

1. A ADAMS, S. S. AND NABI, A. K. NeuralAgents: A Frame of Mind. *OOPSLA '89 Proceedings*, (Oct. 1-6, 1989), pp. 139-149.

2. W AHLSEN, M., BJORNERSTEDT, A., BRITTS, S., HULTEN, C., AND SODERLUND, L. An Architecture for Object Management in OIS, *ACM Transactions on Office Information Systems*, Vol. 2, No. 3, (July 1984), pp. 173-196.

3. A ANDERSON, R. H. AND GILLOGLY, J. J. Rand Intelligent Terminal Agent (RITA): Design Philosophy. ARPA Order No. 189-1, Rand, Santa Monica, CA, (Feb. 1976).

4. A Apple Computer, Inc. Project 2000 — A Knowledge Navigator. (videotape), (Mar. 8, 1988).

5. W BEYOND INC. BeyondMail Rule Book, Cambridge MA, (1991).

6. W BRACCHI, G. AND PERNICI, B. The Design Requirements of Office Systems, *ACM Transactions on Office Information Systems*, Vol. 2, No. 2, (April 1984), pp. 151-170.

7. G CC:MAIL, *LAN*, Miller Freeman Publications (Jan. 1991).

8. A CHEN, F. F., PRAKASH, A., AND RAMAMOORTHY, C. V. The Network Event Manager. *Proceedings of the Computer Networking Symposium*, Washington D.C., (1986).

9. G COOMBS, J.H. Hypertext, Full Text, and Automatic Linking, *SIGIR '90 Proceedings: 13th International Conference on Research and Development in Information Retrieval*, Brussels, Belgium, (September 5-7, 1990) ACM, New York, (1990), pp. 83-98.

10. A DIALOG INFORMATION SERVICES, INC. AP News Reloaded as a Single File; New Fields and DIALOG Alert Service Added. *Chronology*, Vol. 18, No. 4, (Apr. 1990).

11. W BLOC Development Corp. F3 Design & Mapping. Reference Guide, Deerfield Beach, FL (1991).

12. A DON, A., OREN, T., AND LAUREL, B. Guides 3.0. *CHI '91 Video Proceedings*, New Orleans, LI (April 28-May 2, 1991) ACM, New York, (1991).

13. A DOW JONES AND COMPANY, INC. An Overview of DowVision. (1990).

14. A DROMS, R. E. Access to Heterogeneous Directory Services. *Proceedings of the IEEE InfoCOM '90 Conference*, (Jun. 1990).

15. G ELLIS, C. A, GIBBS, S. J., AND REIN, G. L. Groupware: Some Issues and Experiences. *Communications of the ACM*, Vol. 34, No. 1, (Jan. 1991), pp. 38-58.

16. W ELLIS, C. A, NUTT, G. J. Office information Systems and Computer Science, *ACM Computing Surveys*, Vol. 12, No. 1 (March 1980), pp.27-60.

17. A ELLMAN, T. Explanation-Based Learning: A Survey of Programs and Perspectives. *ACM Computing Surveys*, Vol. 21, No. 2, (Jun. 1989), pp. 164-221.

18. W ENGLISH, P., ET. AL. An Extensible, Object-Oriented System for Active Documents, *Conference on Electronic Publishing, Document Manipulation and Typography Proceedings*, Gaithersburg, MD, (Sep. 1990), pp.263-276.

19. A FENTON, J. AND BECK, K. Playground: An Object Oriented Simulation System with Agent Rules for Children of All Ages. *OOPSLA '89 Proceedings*, (Oct. 1-6, 1989), pp. 123-137.

20. A FREEDMAN, BETH. Verity Upgrades Topic Text Manager. *PC Week*, (June 12, 1989).

21. G GIBBS, S.J. LIZA: An Extensible Groupware Toolkit. *CHI '89 Conference Proceedings*, Austin, TX (April 30-May 4 1989) ACM, New York, (1989), pp. 29-35.

22. G GRUDIN, J., POLTROCK, S. Computer Supported Cooperative Work and Groupware, *Conference on Computer-Human Interaction* Tutorial Notes, (Apr. 2 1990)

23. W HAMMAINEN H., ELORANTA E., AND ALASUVONTO J. Distributed Form Management. *ACM Transactions on Office Information Systems*, Vol 8, No 1, (Jan. 1990), pp. 50-76.

24. W HAMMAINEN H., Form-based Approach to Distributed Cooperative Work. Acta Polytechnica Scandinavica, Mathematics and Computer Science Series No. 58, Helsinki (1991).

25. W Nammer, M., Howe, W. G., Kruskal, V. J., and Wladawsky, I. A Very High Level Programming Language for Data Processing Applications, Communications of the ACM, Vol. 20, No. 11, (Nov. 1977), pp. 832-840.

26. A HEWLETT PACKARD, HP NewWave Agent Guide, Santa Clara, CA, (Oct. 1989).

27. W HIRSCHHEIM, R. The Effect of A Priori Views on the Social Implications of Computing: The Case of Office Automation, *ACM Computing Surveys*, Vol. 18, No. 2, (Jun. 1986), pp. 165-195.

28. A KAHN R. E. AND CERF V. G. "The Digital Library Project: The World of Knowbots," Vol. 1, Corporation for National Research Initiatives, (Mar. 1988).

29. W KARBE, N., RAMSPERGER, N., WEISS, P. Support of Cooperative Work by Electronic Circulation Folders, *Conference on Office Information Systems Proceedings*, (1990), pp. 109-117.

30. W,A KAYE, A. R. AND KARAM, G. M. Cooperating Knowledge-Based Assistants for the Office. *ACM Transactions on Office Information Systems*, Vol. 5, No. 4, (Oct. 1987), pp. 297-326.

31. W KEYFILE, INC. Keyfile Documentation(under non-disclosure), Nashua NH, (May 1991).

32. G KRAEMER, K., KING, J. Computer-Based Systems for Cooperative Work and Group Decision Making, *ACM Computing Surveys*, Vol. 20, No. 2, (Jun. 1988), pp. 115-146.

33. G LAI, K. Y. AND MALONE, T. W. Object Lens: A "Spreadsheet" for Cooperative Work. *CSCW '88*, Portland, Oregon, (Sep. 26-28, 1988), pp. 115-124.

34. A LAUREL, B. Interface Agents: Metaphors with Character. *The Art of Human-Computer Interface Design*, ed. by Brenda Laurel, Addison-Wesley Publishing Co., Inc., Reading, MA, (1990), pp. 355-366.

35. W LEWIS, J., BURTON, C. Workflow Automation. Clarke Burton Report, Salt Lake City, UT, Vol. 1, No. 12, (Apr. 1991).

36. W LYYTINEN, K. Different Perspectives on Information Systems: Problems and Solutions, *ACM Computing Surveys*, Vol. 19., No. 1, (Mar. 1987), pp. 5-46.

37. G MALONE, T. W., GRANT, K. R., LAI, K. Y., RAO, R., AND ROSENBLITT, D. Semi Structured Messages are Surprisingly Useful for Computer-Supported Coordination. *ACM Transactions on Office Information Systems*, Vol. 5, No. 2, (Apr. 1987), pp. 115-131.

38. G MARSHAK, D. Lotus Notes: A Platform for Developing Workgroup Applications. *Patricia Seybold's Office Computing Report*, Vol. 13, No. 7, (Jul. 1990), pp.1-12.

39. W MAZER, M. S. AND LOCHOVSKY, F. H. Logical Routing Specification in Office Information Systems, *ACM Transactions on Office Information Systems*, Vol. 2, No. 4, (October 1984), pp. 303-330.

40. G MICROSOFT CORP. Microsoft Mail for PC Networks, Strategic White Paper. Redmond, WA, Part No. 098-19609.

41. G MILEY, M. The Medium is Not the Message. *Personal Workstation*, (May 1991), pp. 49-53.

42. A MINSKY, M. *The Society of Mind*, Simon and Schuster, New York, (1986).

43. A OBJECT MANAGEMENT GROUP. Object Management Architecture Guide. Framingham, MA, (1991).

44. A OREN, T., SALOMON, G., KREITMAN, K., AND DON, A. Guides: Characterizing the Interface. *The Art of Human-Computer Interface Design*, ed. by Brenda Laurel, Addison-Wesley Publishing Co., Inc., Reading, MA, (1990), pp. 367-381.

45. A PALANIAPPAN, M., YANKELOVICH, N., FITZMAURICE, G., ET. AL. The Envoy Framework: An Open Architecture For Agents, To appear in ACM TOOIS (1991).

46. W SPANBAUER, S. Perfect Forms with Windows, PC World, (Jul. 1991), pp. 199-207.

47. A Sun Microsystems Inc. SunNet Manager Installation and User's Guide, Part Number: 800-3481-10, Mountain View, CA, (Mar. 1990).

48. W TSICHRITZIS, D. Form Management, *Communications of the ACM*, Vol. 25, No. 7, (July 1982), pp. 453-478.

49. G TUROFF, M. Computer-Mediated Communication Requirements for Group Support, *Journal of Organizational Computing*, Vol. 1, (1991), pp. 85-113.

50. G WEYER, S. A. AND BORNING, A. H. A Prototype Electronic Encyclopedia. *ACM Transactions on Office Information Systems*, Vol. 3, No. 1, (Jan. 1985), pp. 63-88.

51. W YAO, S. B., HEVNER, A. R., SHI, Z., AND LUO, D. Formanager: An Office Forms Management System, *ACM Transactions on Office Information Systems*, Vol. 2, No. 3, (July 1984), pp. 235-262.

7 Appendices

7.1 Appendix: Logical Routing Evaluation

This system was extensively reviewed in the Background section but since it has so much intersection with workflow models that deal with routing, a more detailed analysis is undertaken. The primary distinction between the systems is that the logical routing system is based on messages or forms that work in isolation. In our workflow model, we envision functionality that requires forms to interact with one another. Such functionality includes:

- Hot linking of data between fields in a form and groups of fields
- Conditional routing based on state information and field values that span multiple forms and possibly all forms within a form set. This entails being able to access and set field values from any form in the workflow.
- Multiple forms displayed during a workflow session (processing and filling out field values).
- Splitting a form instance and allowing each of the split forms to have independent routing until the workflow dictates that the forms be merged and reconciled.

The Message Management System cannot easily support this level of functionality and breaks down for the following reasons:

- The routing language has been designed to route individual, autonomous messages independent of any other messages. It is difficult for designers to access field values belonging to message instances of the same type or of different types all together.
- The concurrency and access control for entire message instances and field values within messages is left to the communication base, implemented as a relational database. The routing language has no constructs for the form designers to access and set this type of information.
- Some level of tracking information is kept within the communication base but no applications exist to display and summarize this information. It is anticipated that tracking applications could be built which hook into the communication base.
- Having distributed routing evaluation but relying on a centralized communication base is problematic in terms of performance and being vulnerable to having the entire system become inoperable if the communication base is off-line. Using

multiple, decentralized communication base may be more reliable but makes it more difficult to perform functions that span more than one communication base.

- Distributing the message type definitions and routing script based on mapping a role (e.g., grad-coordinator) to a site machine is problematic if roles changes frequently or if users work on different machines. Especially in a UNIX environment, users often log into more than one workstation to do their everyday work; this means that the site of the user (or role) is often not static and not known before the message instance is launched.
- Splitting a form into "originals" is possible but the added restriction of requiring all originals to be merged and routed to the same next site is unnecessary and prevents flexibility in the design of some workflows.

Our approach has many advantages over the MMS in that the design allows for an easier implementation of the desired form-centered workflow functionality that works across multiple forms.

- Updating each workstation with the necessary message type definitions as well as ensuring that the latest version of each definition is available is a cumbersome process in the MMS. In our agent-base workflow model, having centralized form-specific servers obviates the problem of updating each workstation with the necessary form class definitions.
- Roles are resolved at the last possible moment by the Bureau Chief. Forms are accessed by making requests to a server running at an advertised machine and port number. The Form Frontend can even run remotely if the local site does not have a copy of the executable (since we are using the X Windowing System). This means that users can move around to different workstations and still access and process their workflows.
- Each form server makes use of a database to provide data persistence and consistency.
- Our routing library allows designers to access and set field values from any form in a given workflow schema (even between different form classes).
- The state information of the workflow instance is centrally located in the Workflow Envoy and is accessible from any form and its related routing code.

The MMS allows Message designers the ability to alert users when a message has not been processed in a timely manner or can warn users when a certain duration has passed. The notification is in the form of an alert box. This primitive notification is not sufficient in an environment when many messages may be active; the user could get bombarded with a relentless series of alert box interruptions. A more comprehensive monitoring and management tool needs to be provided. These abilities fall under the general tracking functionality that must be addressed more vigorously in form automation systems.

The paper ends with issues that need to be addressed in the future. One problem involves dealing with changing message type definitions. If message instances of a given type are still active while a new message type definition is installed, what happens? A similar problem occurs when roles change within an organization.

Finally, the paper does not aggressively target just office information systems but also hints at applying the logical routing specification to networks and transport mechanisms in general.

7.2 Appendix: Envoy Framework Overview

IRIS has taken the approach that agents should be desktop-based, personal assistants that operate in conjunction with user's existing set of applications. Each user has one agent, or Envoy, assigned to them to work on their behalf. APIs have been designed so that developers can upgrade their software into an envoy-aware application. The Envoy serves as a delegator and manager of task assignments and results. In its current design, Envoys are not responsible for carrying out action for the user; this is left for the applications. The Envoy aids in keeping track of the user's set of tasks and informs the user when a task is completed by using a variety of notification channels (e.g., e-mail, alert box).

We have adopted a playful metaphor to help users understand and interact effectively with Envoys. A user specifies a *mission* for his Envoy by interacting with an "envoy-aware" application. We call envoy-aware applications *operatives* because they are responsible for actually carrying out missions on behalf of the user. Once the user specifies a mission, the Envoy plays the role of coordinator, scheduling, tracking, and dispatching all missions the user has specified (Figure A).

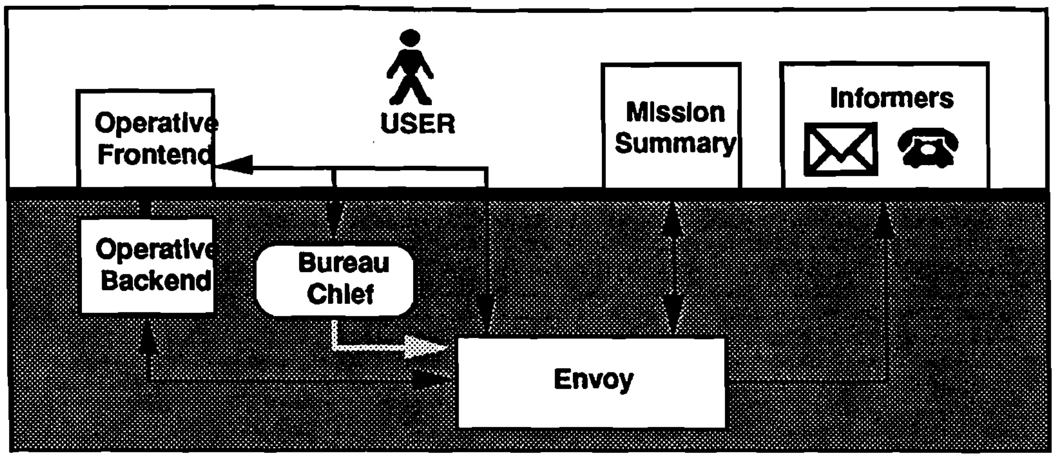


Figure A: Overview of Envoy Framework components.

The Envoy handles all communication with the operatives. If the user has specified an information-gathering mission, then the operative assigned to the mission lets the Envoy know when new information is available to report to the user. In turn, the Envoy notifies the user, selecting a communication channel from a set of envoy-aware applications called *informers*. Once notified of mission results, either with a brief *message* or a *short report*, the user can opt to see an *interactive report*. The Envoy stores interactive reports generated by operatives. To view an interactive report, the Envoy passes the data to the operative responsible for carrying out the mission, giving users the ability to manipulate the mission results using the native application interface. At any time, the user may display a *Mission Summary* which provides a comprehensive list of all the user's active missions and all the reports generated by the operatives responsible for those missions.

When an application developer first introduces a new operative or informer into the environment, she registers the application with a *Bureau Chief*. For every local-area network, there is one Bureau Chief which maintains a record of all envoy-aware applications in the environment as well as a record of each user's personal Envoy.

7.3 Appendix: Implementation Schedule

This section describes the implementation plan for building the workflow prototype based on the Envoy framework. A two phase plan was defined to ensure that a minimum workflow model would be working within two and a half weeks (i.e., Phase I). Milestones and their completion data were given to ensure successful progress. Note that each milestone was met on the specified days. This can be considered proof that the Envoy framework was extensible for this application.

Milestone 0 (Sat. 3th)

Install a new IRIS build & development tree. The environment will consist of C++, X11 and motif. IRIS' list building block will also be used.

Phase I

First, a Form operative Frontend base class and executable will be built along with a sample form. The Frontend will provide form presentation, read/write protection on fields and field fill in. The interface will be ASCII based. A command loop will prompt the user to hit a character for issuing a form-related command.

The Frontend needs to be able to receive a partially completed form (i.e., a data packet that contains field values) and authorization codes. The code will consist of one byte per field containing access and status information: read, write, visible, hot-linked, fill-in requested, already filled-in.

Milestone 1 (Tue. 6th)

Issue a command that launches the form Frontend. User should be able to display the form, fill-in the form and the authorization information should be enforced.

The next goal is to concentrate on providing the communication mechanism between the various components. This requires that the Backend, Workflow Envoy and User Envoy all be initially built. The idea is to build a skeleton server for each component that can contact, send and receive messages from one another. TCP/IP will be used for the network protocol. RPC was considered but it is too cumbersome to use and dynamically change.

Once the communication mechanisms are in place, the Backend of the form operative will be enhanced next. It will contain minimum functionality such as the core dispatching routine and the routing state table.

Milestone 2 (Mon 12)

The next milestone will be to successfully execute the following scenario: (1) issue a command that launches a workflow instance.(2) This causes the WE to contact a form Backend which creates a new form instance. (3) Next the User Envoy is contacted with the work request and (4) the form Frontend is launched automatically.

Next the FRforward() calls will be implemented and tested to work for linear, non-splitting workflows that involve a single form. This requires that the role-procedures be accessible from the state table.

The transaction log should be built next for the WE. At first, ASCII based debugging information (e.g., fprintf()'s) will be used for the Workflow Envoy. They will print out the transaction log. The WE will need to contact users' User Envoy and install the work request. The User Envoys will need to be able to launch a form Frontend. The Frontend needs to be able to packetize the form data and send it to the WE.

Milestone 3 (Fri. 16)
Initiate a workflow instance and have the single form be filled out by two or three users (see Testing). The WE should be keeping track of each message transaction. The User Envoys need to show the work requests in the Mission Summary application and be able to access the work request (e.g., launch it).

Note that in choosing an ascii-based Frontend interface, it is not readily possible to launch multiple forms during the workflow stage initiation. (One possible solution is to launch an xterm window per form that needs to be displayed. We could automatically launch the appropriate Frontend executable in each xterm window – this needs to be investigated).

If it is possible to launch multiple forms during a workflow stage, then the form set data structure will be defined to provide this ability.

Milestone 4 (Sat. 17)
Define and test a workflow instance that requires two forms (of the same type) to be displayed at a given workflow stage.

Define more complicated workflow schemas that involve splits and joins and multiple form types. The Backend and WE will need some additional enhancements. The idea is to provide inter-form communication by implementing the FRgetFieldData() and FRsetFieldData() calls. FRspawn(), FRsuspendMerge() and FRsuspendFormArrival() will also be implemented.

Milestone 5 (Fri. 23)
Execute workflows that require a form to be spawned, split and merged. Show conditional routing and cyclic routing examples. Show data access and retrieval between forms.

Phase II Implementation

Time permitting, each of the items in the Phase II list will be built.

The Bureau Chief will be one of the last components built. A role resolution server can be provided by a simple ASCII data file. The remaining Bureau Chief functionality is secondary and can be "hard-coded" and later implemented if time permits..

Hot linking data fields within a form set will be implemented after the basic routing abilities are working.

Hook-up the GUI Mission Summary application to the workflow envoy. The Mission Summary application, which works in conjunction with the User Envoy, will be significantly modified and attached to the Workflow Envoy to display status information about various workflow instances.

Provide workflow instance and form instance persistence.

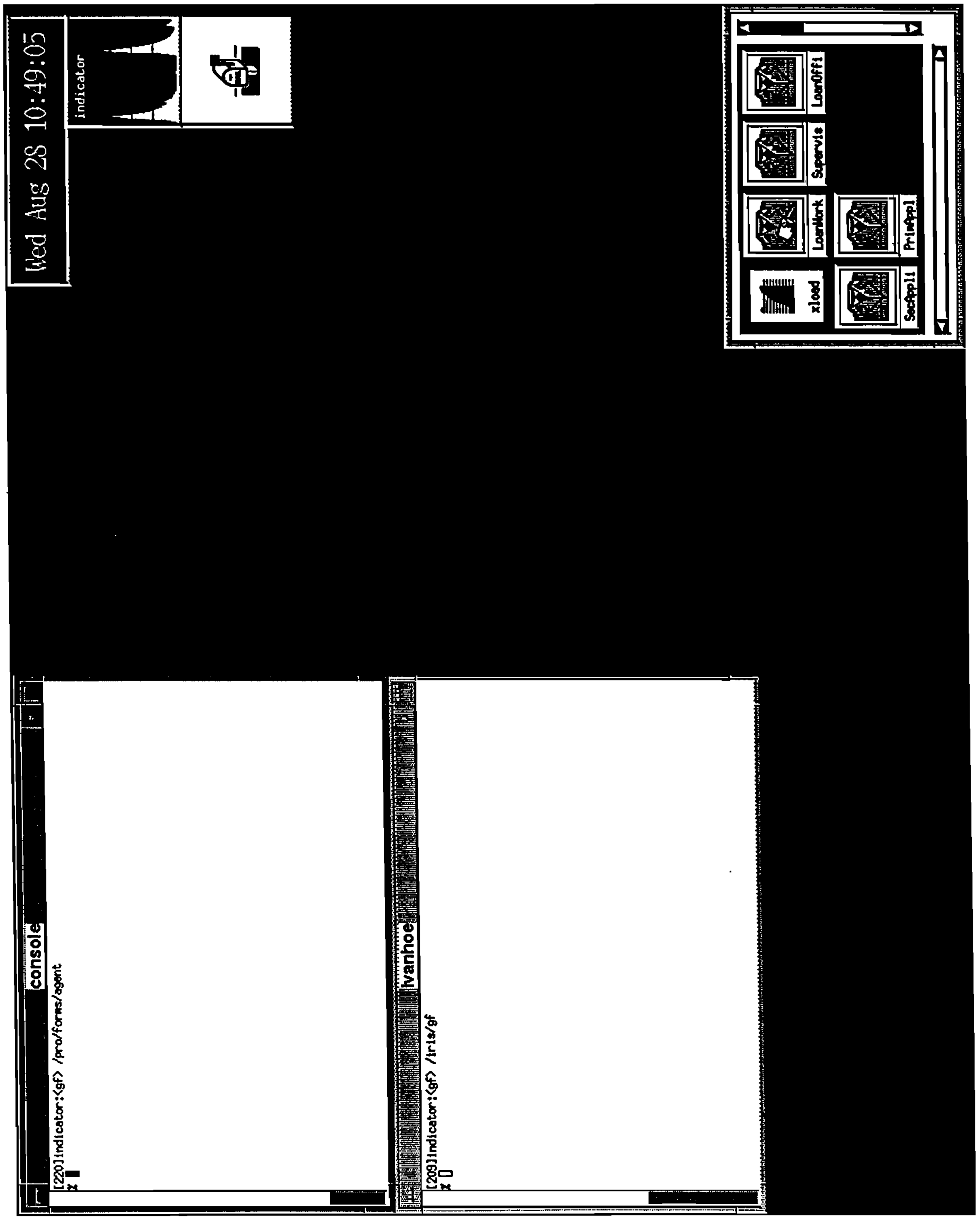
Experiment with tracking data and summary statistics. Resource balancing.

Testing

In order to test the system, the User Envoys will access environment variables to retrieve the userID. This allows a single user to log into multiple windows and, by setting an environment variable, appear to be many people having different userIDs. Consequently, the single user will be able to display a set of Mission Summaries corresponding to each virtual user. This will allow the tester to see and interact with each stage in a given workflow instance.

7.4 Appendix: Prototype Screen Shots

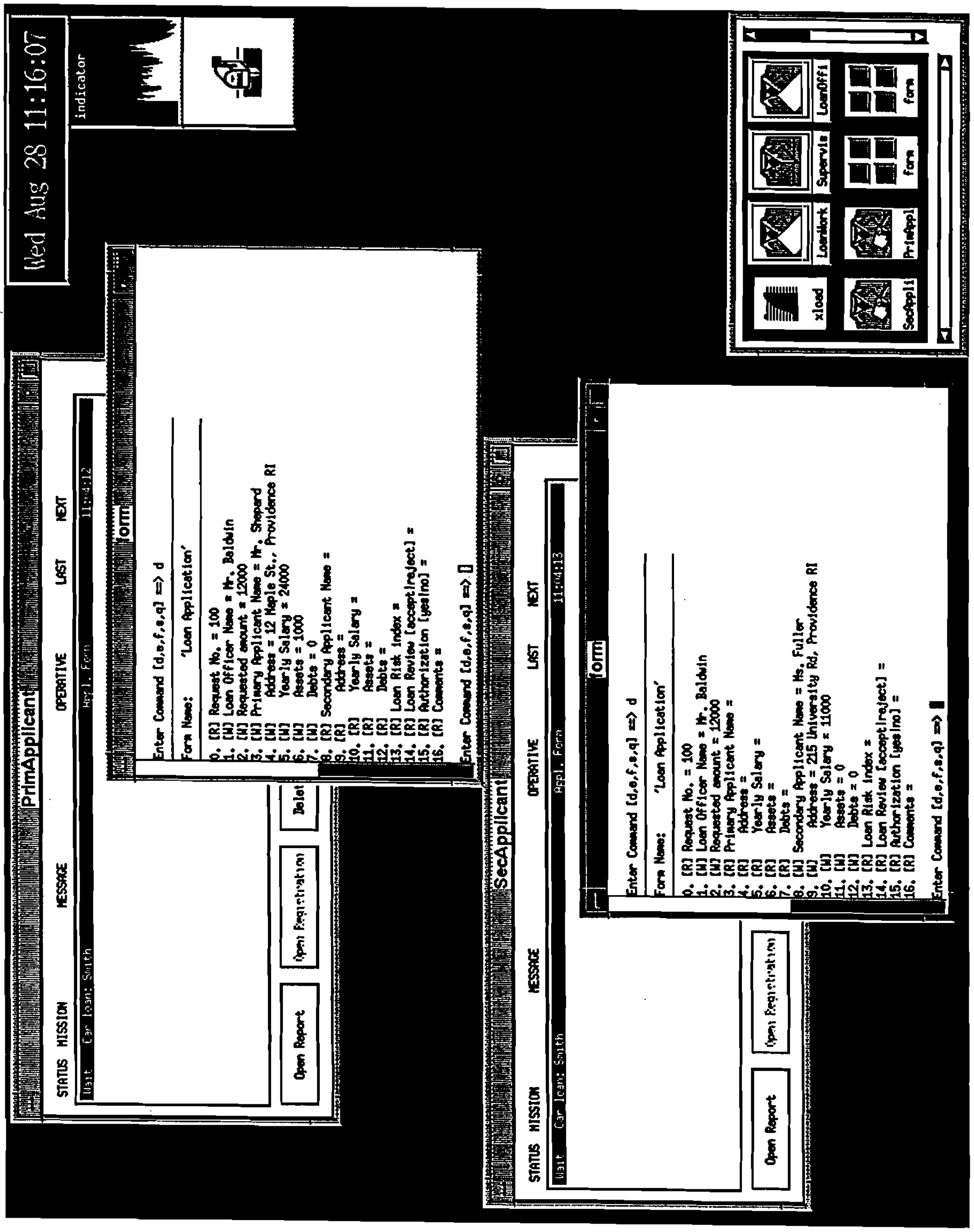
Figure 8. Five Envoy Icons, the initial workflow state.



```
[220]indicator:<gf> /pro/forms/agent
% 
```

```
[209]indicator:<gf> /iris/gf
% 
```

Figure 9. Splitting the application for both of the applicants. Each applicant can fill-in the form in parallel.



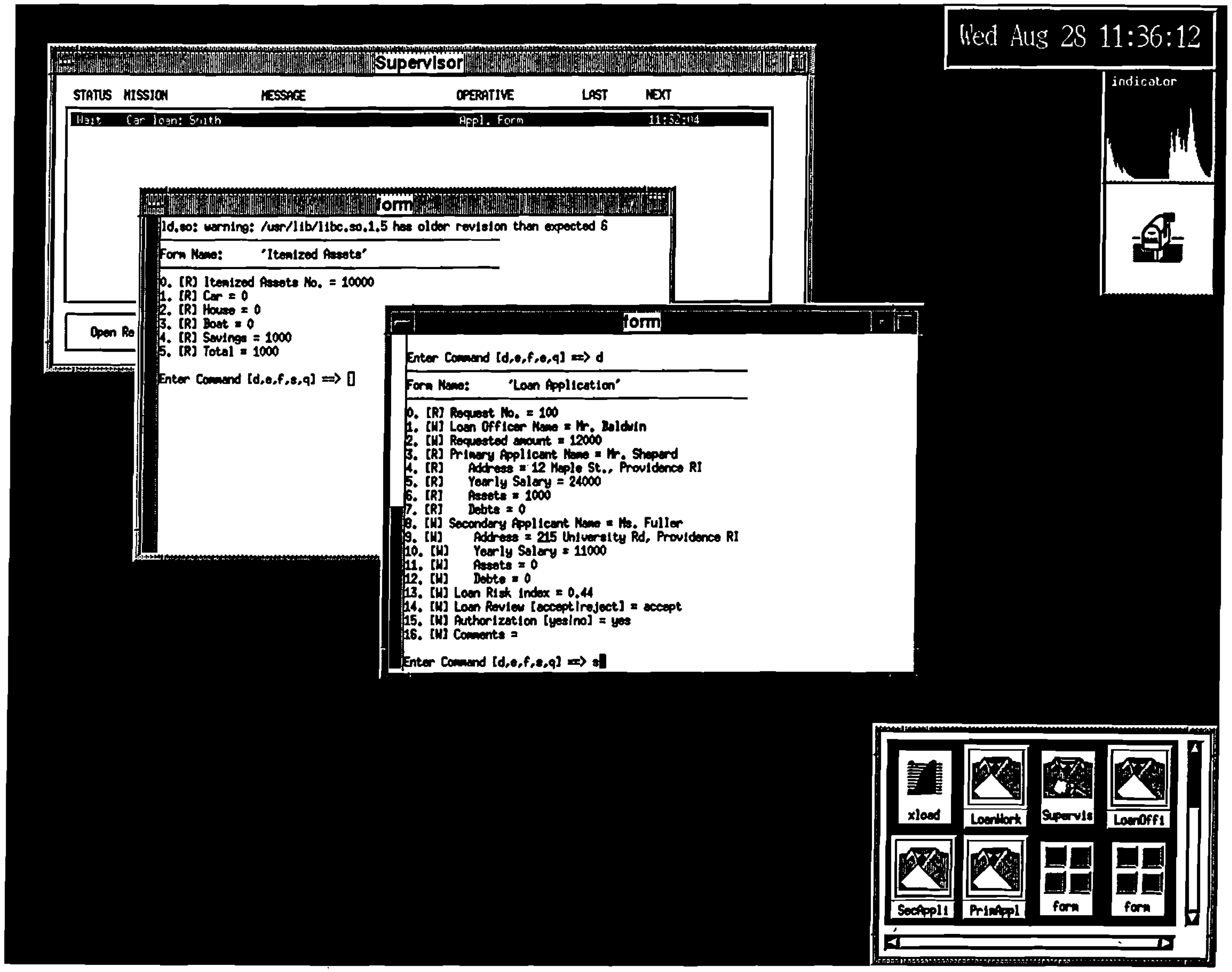
```
STATUS  MISSION          MESSAGE          OPERATIVE    LAST    NEXT
Wait    Car loan: Smith                   Appl. Form           11:04:12
```

```
Enter Command [d,e,f,s,q] ==> d
Form Name:      'Loan Application'
0. [R] Request No. = 100
1. [W] Loan Officer Name = Mr. Baldwin
2. [W] Requested amount = 12000
3. [W] Primary Applicant Name = Mr. Shepard
4. [W]     Address = 12 Maple St., Providence RI
5. [W]     Yearly Salary = 24000
6. [W]     Assets = 1000
7. [W]     Debts = 0
8. [R] Secondary Applicant Name =
9. [R]     Address =
10. [R]    Yearly Salary =
11. [R]    Assets =
12. [R]    Debts =
13. [R] Loan Risk index =
14. [R] Loan Review [accept|reject] =
15. [R] Authorization [yes|no] =
16. [R] Comments =
Enter Command [d,e,f,s,q] ==> 
```

```
STATUS  MISSION          MESSAGE          OPERATIVE    LAST    NEXT
Wait    Car loan: Smith                   Appl. Form           11:04:13
```

```
Enter Command [d,e,f,s,q] ==> d
Form Name:      'Loan Application'
0. [R] Request No. = 100
1. [W] Loan Officer Name = Mr. Baldwin
2. [W] Requested amount = 12000
3. [R] Primary Applicant Name =
4. [R]     Address =
5. [R]     Yearly Salary =
6. [R]     Assets =
7. [R]     Debts =
8. [W] Secondary Applicant Name = Ms. Fuller
9. [W]     Address = 215 University Rd, Providence RI
10. [W]    Yearly Salary = 11000
11. [W]    Assets = 0
12. [W]    Debts = 0
13. [R] Loan Risk index =
14. [R] Loan Review [accept|reject] =
15. [R] Authorization [yes|no] =
16. [R] Comments =
Enter Command [d,e,f,s,q] ==> 
```

Figure 10. Final review of the workflow by the Supervisor. Note that all forms belonging to the workflow are displayed at this workflow stage.



```
STATUS  MISSION          MESSAGE        OPERATIVE      LAST     NEXT
Wait    Car loan: Smith                 Appl. Form              11:32:04
```

```
ld.so: warning: /usr/lib/libc.so.1.5 has older revision than expected 6

Form Name:     'Itemized Assets'

0. [R] Itemized Assets No. = 10000
1. [R] Car = 0
2. [R] House = 0
3. [R] Boat = 0
4. [R] Savings = 1000
5. [R] Total = 1000

Enter Command [d,e,f,s,q] ==> []
```

```
Enter Command [d,e,f,s,q] ==> d

Form Name:     'Loan Application'

0. [R] Request No. = 100
1. [W] Loan Officer Name = Mr. Baldwin
2. [W] Requested amount = 12000
3. [R] Primary Applicant Name = Mr. Shepard
4. [R]    Address = 12 Maple St., Providence RI
5. [R]    Yearly Salary = 24000
6. [R]    Assets = 1000
7. [R]    Debts = 0
8. [W] Secondary Applicant Name = Ms. Fuller
9. [W]     Address = 215 University Rd, Providence RI
10. [W]    Yearly Salary = 11000
11. [W]    Assets = 0
12. [W]    Debts = 0
13. [W] Loan Risk Index = 0.44
14. [W] Loan Review [accept|reject] = accept
15. [W] Authorization [yes|no] = yes
16. [W] Comments =

Enter Command [d,e,f,s,q] ==> s
```

7.5 Appendix: Visual Routing Language Scenarios

We provide three scenarios that any workflow system should be able to handle. After describing the scenarios, a visual diagram, or flow chart is presented that graphically represents the scenarios in an iconic fashion. This is preliminary research on a more robust and extensive visual language specification.

Scenario 1: LINEAR — Student registration for class

An employee at Brown wishes to register for a class. He first goes to the Brown Learning Community which requests that an application form be on file before moving to the next step. A packet of information is then received. The packet contains a registration form.

Since registration happens for only one more day, this must be attended to first. Upon waiting in line at the Registration office the employee must fill out two additional forms which ask for routine information such as your name, student id, address, parent's address, telephone numbers, etc. Note that if the employee is formally enrolled as a degree candidate, the registration form would require the authorization of his advisor (or any faculty belonging to the department) before it travels to the Registra. If the advisor does not agree with the registration, s/he needs to discuss the problem with the student before authorizing the form. A copy of the completed registration form is given to the student.

Once the registration form is submitted, the tuition reimbursement form can be filled out. The form requires the employee to fill out standard information and specify which course he wants to take and why it is job related. Next, he gives the form to his supervisor who authorizes the request. The supervisor accepts or rejects the request. The form next travels to the benefits office which will process and authorize the request. Before requesting for tuition reimbursement, the benefits office must receive a copy of the employee's official grade for any classes taken the previous semester. If a student receives a grade lower than a C, he does not receive total reimbursement but must pay the auditing fee for regular Brown courses (or 80% of tuition for BLC courses taken). Before requesting reimbursement, the employee must also authorize the Payroll department at Brown to deduct $100 per month from his salary if it becomes necessary to pay for a course. Note that the tuition reimbursement form has 5 copies: one for the employee, one for his supervisor, and 3 for the benefits office. Also note that, ideally, the tuition reimbursement process should be completed before the date in which course registration can be changed without penalty.

Scenario 2: CYCLIC — College Admissions cycle.

An admissions committee for a college makes use of a cyclic admissions process. When a prospective student application arrives at the admissions office, it is reviewed by any one of the workers at a first pass to see if the application is complete (all the required material has been received by the committee). If the application is not complete, the file gets placed in a suspend pile while it waits for the remaining material.

Once a file is complete, a first pass review of the application occurs. This first pass checks for minimum requirements and accuracy in filling-out the application forms. Minimum requirements may be GPA > 2.0 and GRE scores totaling more than 1000. The file may get processed sequentially by two workers just to make sure that no errors occur. The application now gets suspended until the application admissions deadline arrives. This allows the committee to know how many total applications it can consider along with how many slots are opened for the upcoming year.

The next round of review evaluates an applicant's writing ability. Three committee members sequentially read a sample of writing and determine the applicant's writing ability. Two of the three must accept the application in order for it to advance to the next review cycle.

Letters of recommendation are considered next. Here the committee is looking for unsupportive recommendations that would mandate a rejection or outstanding recommendations that would support acceptance.

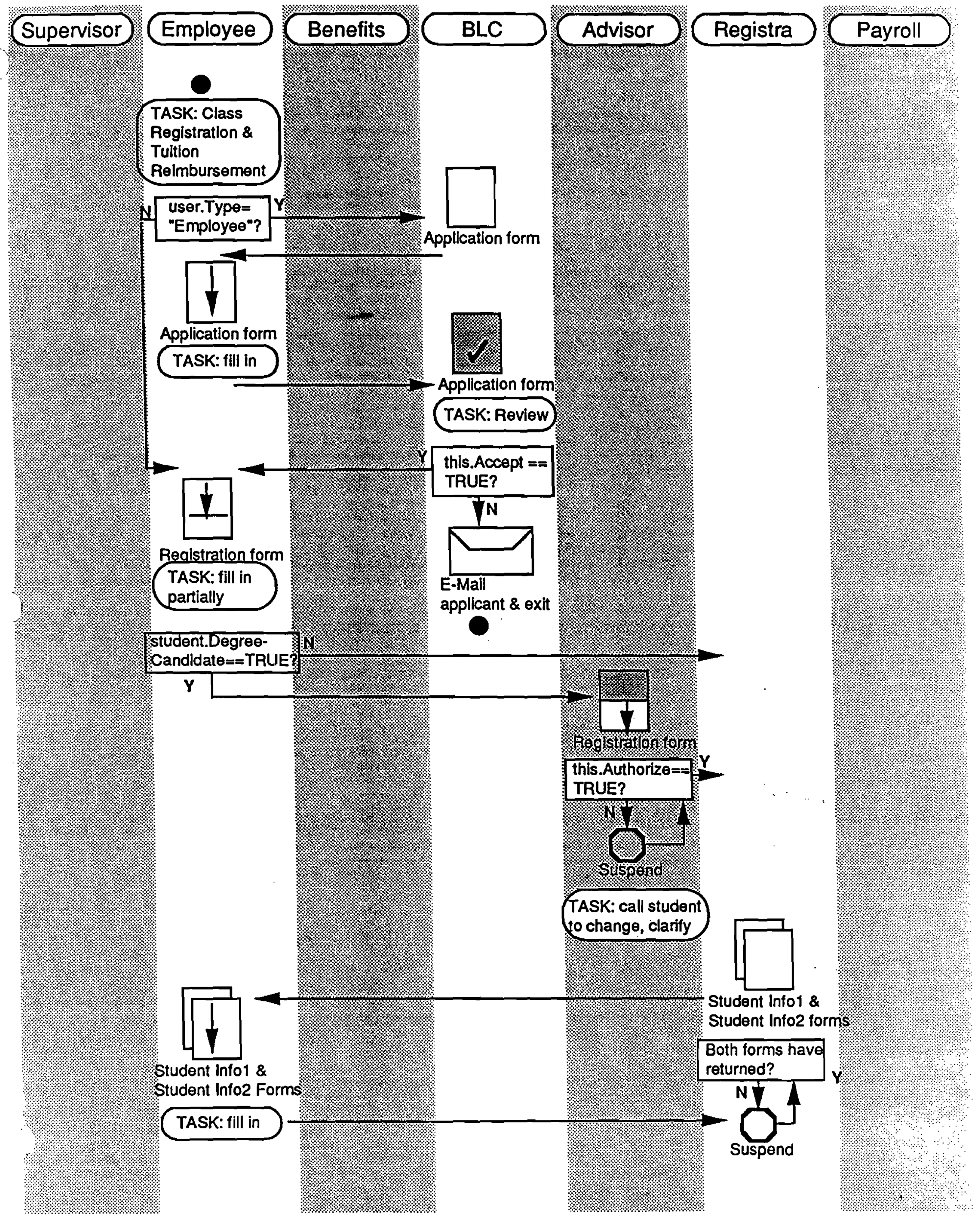
The final review cycle involves the most subjective component in which the committee attempts to look at the remaining applications and identify any unique characteristics (say, in the extra-curricular activities) or indicators that the applicant will succeed at the college. Four out of five committee members must agree upon an a decision of accept, reject, or wait-listed.

Note that all the evaluation constraints cannot be placed on the application in a one pass review. If this approach was chosen, the committee may not make enough offers and then it would be difficult to decide which of the rejected applicants to consider. Each round of review is a refinement process.

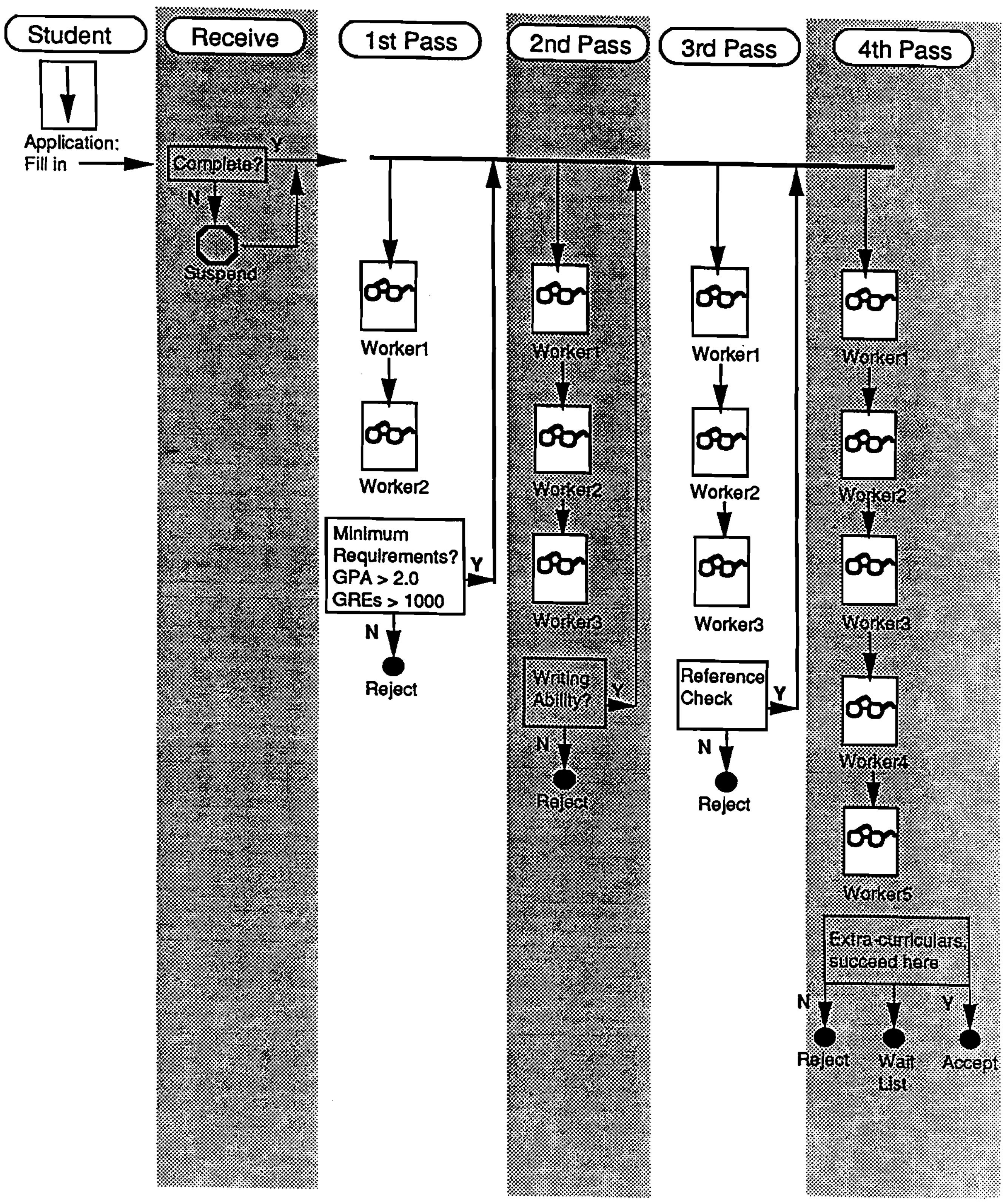
Scenario 3: REPLICATE & CONSOLIDATE — Loan approval.

Two brothers, Tom and Jack, apply for a mortgage for a house at a local bank. Both of them fill out a separate application form. The bank begins to process the application by first using a single form which holds review information concerning the request. Each applicant's information is verified in parallel. The verification process involves three bank workers each responsible for verifying a portion of the application: (1) credit check – long term debts and payment history, (2) asset check – major assets like cars, boats, other property as well as salary, and (2) references. Since two people are applying for the loan, a total of six bank workers can process parts of the form at the same time. When they are done, each component must be consolidated and a risk number is calculated. The loan is accepted or rejected based on the risk number.

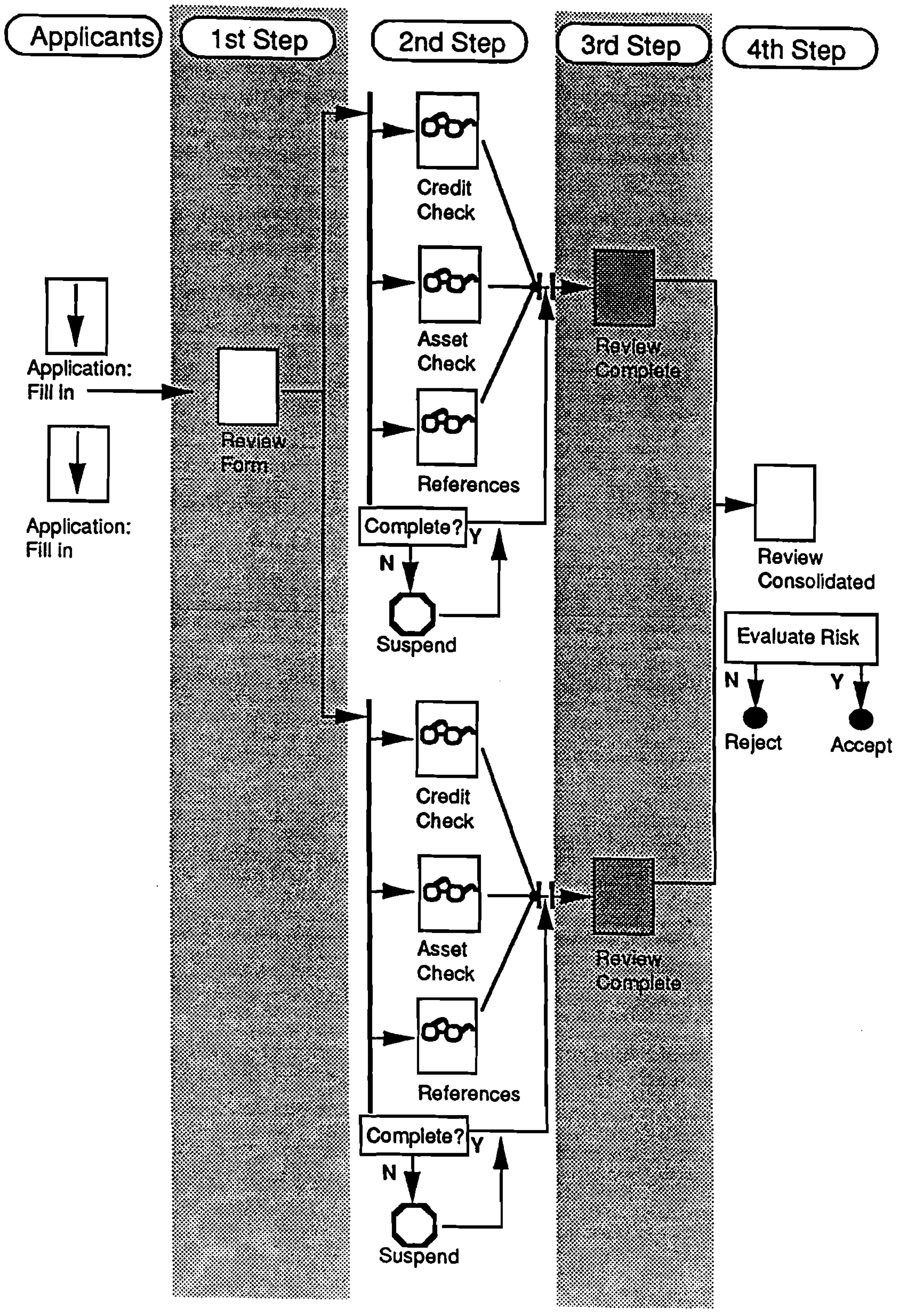
Scenario 1: Linear – Student Registration & Reimbursement

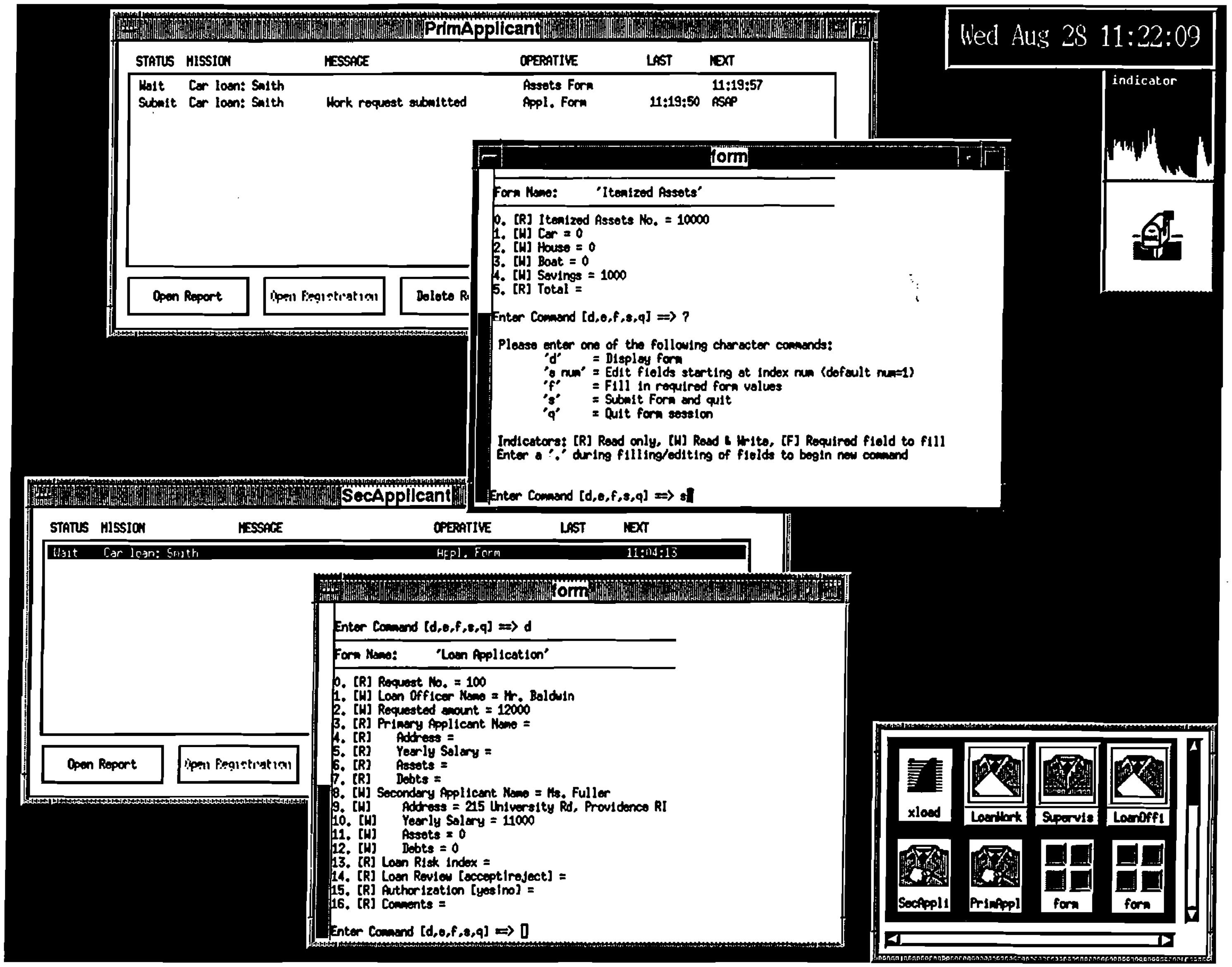


Scenario 2: Cyclic – College Admissions Cycle



Scenario 3: Replicate & Consolidate – Loan Approval





```
STATUS  MISSION           MESSAGE                  OPERATIVE    LAST      NEXT
Wait    Car loan: Smith                            Assets Form            11:19:57
Submit  Car loan: Smith   Work request submitted   Appl. Form   11:19:50  ASAP
```

```
Form Name:      'Itemized Assets'

0. [R] Itemized Assets No. = 10000
1. [W] Car = 0
2. [W] House = 0
3. [W] Boat = 0
4. [W] Savings = 1000
5. [R] Total =

Enter Command [d,e,f,s,q] ==> ?

 Please enter one of the following character commands:
        'd'     = Display form
        'e num' = Edit fields starting at index num (default num=1)
        'f'     = Fill in required form values
        's'     = Submit Form and quit
        'q'     = Quit form session

 Indicators: [R] Read only, [W] Read & Write, [F] Required field to fill
 Enter a '.' during filling/editing of fields to begin new command


Enter Command [d,e,f,s,q] ==> s
```

```
STATUS  MISSION           MESSAGE                  OPERATIVE    LAST      NEXT
Wait    Car loan: Smith                            Appl. Form             11:04:13
```

```
Enter Command [d,e,f,s,q] ==> d

Form Name:      'Loan Application'

0. [R] Request No. = 100
1. [W] Loan Officer Name = Mr. Baldwin
2. [W] Requested amount = 12000
3. [R] Primary Applicant Name =
4. [R]    Address =
5. [R]    Yearly Salary =
6. [R]    Assets =
7. [R]    Debts =
8. [W] Secondary Applicant Name = Ms. Fuller
9. [W]     Address = 215 University Rd, Providence RI
10. [W]    Yearly Salary = 11000
11. [W]    Assets = 0
12. [W]    Debts = 0
13. [R] Loan Risk index =
14. [R] Loan Review [accept|reject] =
15. [R] Authorization [yes|no] =
16. [R] Comments =

Enter Command [d,e,f,s,q] ==> []
```