5-1-1996

DartFlow: A Workflow Management System on the Web using Transportable Agents

Ting Cai
Dartmouth College

Peter A. Gloor
Dartmouth College

Saurab Nog
Dartmouth College

DARTFLOW: A WORKFLOW MANAGEMENT SYSTEM ON THE WEB USING TRANSPORTABLE AGENTS

Ting Cai, Peter A. Gloor, Saurab Nog
Department of Computer Science
Dartmouth College
Hanover, NH 03755

{tcai,gloor,saurab}@cs.dartmouth.edu

Technical Report PCS-TR96-283

Abstract

Workflow management systems help streamline business processes and increase productivity. This paper describes the design and implementation of the DartFlow workflow management system. DartFlow uses Web-browser embedded Java applets as its front end and transportable agents as the backbone. While Java applets provide a safe and platform independent GUI, the use of transportable agents makes DartFlow highly flexible and scalable. This paper describes the design and implementation of DartFlow, as well as a workflow application that exploits DartFlow's agent-based design.

1. Introduction

Rising costs, international competition, and rapidly changing boundary conditions require fast and flexible adaptation to a quickly evolving business environment. While earlier improvement efforts left the internal business organization intact, business process reengineering (BPR) [Ham92] places its whole emphasis on rebuilding the corporation. As enabling technology for BPR, workflow management has experienced tremendous growth in the last few years. Workflow management deals with the specification and execution of business processes. General workflow specifications include the actions to be performed, routing information and policies that describe the organizational environment. By computationally supporting business processes, workflow management systems reduce reaction time to changing requirements and the emergence of new organizational concepts.

The authors are listed in alphabetical order.

This research is partially supported by ONR contract number N00014-95-1-1204 and AFOSR contract number F49620-93-1-0266.

Workflow management systems allow one to dynamically define, execute, manage, and modify business processes. It is irrelevant whether these are business-critical processes such as opening a bank account, buying insurance, applying for credit, or company-internal processes such as job applications or procurement of new office supplies.

From a software engineering perspective, workflow management systems are particularly attractive: The use of workflow management systems isolates the control flow of an application from the domain-specific business logic. Permanent data and operations on this data, such as bookkeeping, are much more stable than the business process itself. By logically separating the process control flow from the data, it is much easier to modify the business process without having to change the application and data structure. Workflow management systems are also particularly well suited for the integration of existing application modules into new applications. This makes it much easier to reengineer legacy applications so that old and new application components can be integrated easily.

For widespread acceptance of workflow management systems, particularly between different companies, adherence to interface standards is of paramount importance. While the Workflow Management Coalition (WfMC) reference model (Figure 1) tries to define a general framework, it still falls short of assuring interpretability between different workflow management systems.

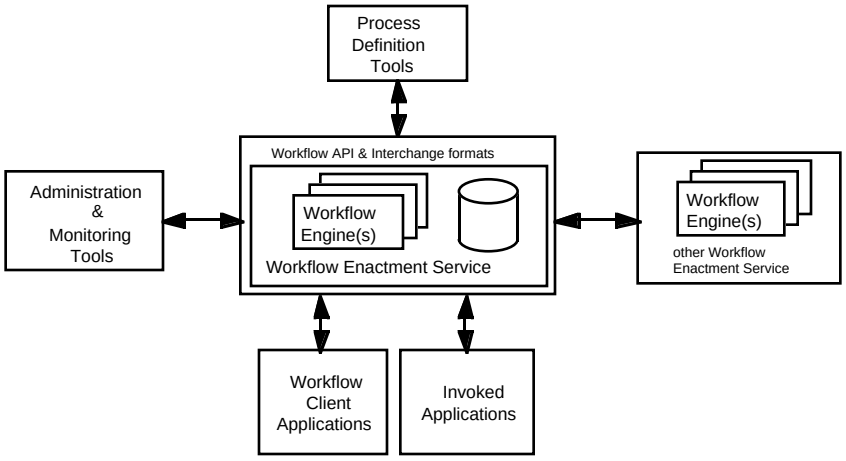


Figure 1 WfMC Reference Model

A different approach to assure interpretability pursued by most workflow vendors today is to use the World Wide Web [Ber92] as the least common denominator. Unfortunately, WWW integration is mostly limited to offering a web browser interface to different proprietary workflow engines. In this paper we propose a different approach using open and portable Web technology not only as a front end for the workflow client applications, but also for the implementation of the workflow enactment service and for administration and monitoring. Section 2 surveys related work and identifies the main

research problems. Sections 3, 4 and 5 introduce the design and implementation of our agent-based DartFlow workflow management system. Section 6 illustrates the use of DartFlow on a sample process, the account opening process for a bank. We conclude by identifying topics for further research.

2. Related Work and Issues Addressed

Although workflow management systems still have a long way to go until they can be considered mature technology, the first generation of commercial systems has started to find wide acceptance. To name a few, systems such as IBM's FlowMark, Action Technology's Action Workflow System, Staffware's Staffware and Wang's OPEN/Workflow [The95], offer usable solutions today. Nevertheless, all of these commercially available systems exhibit common weaknesses. Commercial workflow systems lack transactional properties that have been addressed in database systems research; such as concurrence, availability, performance, scalability, and fault-tolerance [Alo96]. While addressing most of these issues, our work also tackles additional problems that are inherent in the nature of workflow systems:

- Extendibility:
  A few systems such as DEC Linkworks [Dec95] allow the user to dynamically modify a particular process instance by rerouting the work item in the worklist to additional users. But most workflow systems are currently limited to executing canned processes that have been previously defined and are stored in the internal process database of the workflow enactment engine.
- Flexible organization structure:
  Current workflow systems have a built-in database of the organizational structure that defines the recipients for each work item. This mechanism does not scale well for large workflows in large organizations. It is also hard to keep up to date for rapidly changing processes and organizations. There have been some proposed solutions, most prominently language-based approaches such as the one by DEC [Bus94]. Unfortunately most of these approaches currently exist only on paper.
- Structured and unstructured processes:
  Workflow systems are well suited to support structured processes (Figure 2). Unfortunately, complex real-world processes frequently have an unstructured as well as a structured part. For example, a "new product approval process" consists of informally collecting opinions of colleagues as well as a predefined approval process within the company hierarchy. Commercial workflow systems are poorly equipped for handling such semi-structured processes. IBM is working on a possible solution by combining its workflow system FlowMark with the

groupware system Lotus Notes [The95]. Obviously this approach provides only the technological infrastructure; it is still up to the workflow programmer to implement interfaces to unstructured workflows.

	Workgroup	Workflow
Process Type	unstructured (ad-hoc) processes support of teamwork	structured processes productive business processes
Number of Participants per Process	"short" processes in small team few participants per process	company-wide many different participants
User Interaction	Users decide when and what information they want to receive	Users are presented with information on which they have to make decisions
Emphasis	Sharing Information	Routing Information

Figure 2 Workgroup and Workflow Processing

In the remainder of this paper we describe the design and implementation of our agent-based DartFlow system. DartFlow is particularly well suited to address the three problems raised above.

3. Workflow by Agents

Various approaches to workflow management systems have been explored in the past. Most existing systems have been based on transaction-processing systems, extended e-mail systems, or rule-based systems. Our solution to an efficient, flexible and robust workflow management system is to use transportable agents. A transportable agent is a program that can migrate under its own control from machine to machine in a heterogeneous network. In other words, the program can suspend its execution at an arbitrary point, transport to another machine, and resume execution on the new machine. These agents may have certain advantages. First, they may reduce network traffic, because they eliminate intermediate data transfer by moving to the network location of the user or resource. Second, they support systems such as mobile computers that have unreliable or non-permanent network connections, because they can travel into the network and act autonomously even if their home machine, e.g., a laptop, has been disconnected.

Transportable agents are particularly suitable for workflow management systems. Each business process can be handled by an agent. The agent follows the steps in a previously defined process,

migrates to each site and gathers information. While traveling, an agent carries process-specific code and data; there is no need to consult the central database server at every step.

Since each business process instance is controlled by a transportable agent, an agent-based workflow system allows dynamic modification of a particular process. If, for example, an exception occurs during a process, the agent can contact an exception handler server. The rules stored on the server modify that particular process instance without affecting the general process definition. An agent can also acquire knowledge during a business process, such that there is no need to contact the exception handling server again the next time a similar exception occurs. Also, intelligent routing can be implemented efficiently by transportable agents. If one user is busy or absent, the agent can jump to the next candidate.

During a business process, some information from a remote site may be needed, such as the credit history of an applicant from a central credit bureau. Instead of transferring all the data to the requesting site, a small agent can travel to the remote site, eliminating intermediate traffic and improving data integrity, since the data never leaves the repository. In addition, the remote site only needs to provide a primitive query interface. There is no need to build a application-specific high level interface at each remote site, since agents can implement the best interface for themselves and all intermediate communication are local.

Since agents are full programs, they include the potential to explore parallelism. Each process can be regarded as a collection of tasks, the ordering and scheduling of which are subject to dependence constraints. Data dependence exists when tasks produce data that are needed by other tasks. Traditional compiler techniques, such as dependency graphs analysis could be used to identify spurious or nonessential dependencies and executes independent tasks in parallel. For example, in an unstructured process, such as informally collecting opinions of colleagues, several child agents can be created to collect data independently, then join together.

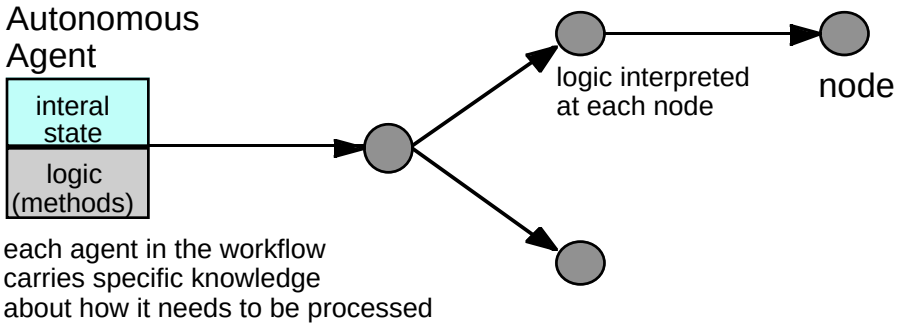


Figure 3 Transportable Agents for Workflow Management

The transportable agent knows where within the workflow process it needs to travel (Figure 3). It also stores some basic knowledge about its internal state and behavior. When an agent arrives at the user's workstation, it acts according to the information it is carrying within itself. The agent stores a process map (routing information) and generic object information that can be used at various steps in the workflow, consisting of state variables to store the application-specific data and methods for the particular object's class. This means that each agent executing a workflow process has detailed knowledge about how to execute its task, i.e., it stores the business rules for the successful execution of each step of the task.

In short, transportable agent-based workflow management systems can be more efficient, flexible and fault tolerant.

4. The Agent Tcl System

For our research we are using a transportable agent system called Agent Tcl. Agent Tcl [Gra95, Gra96] is a system for transportable agents currently being developed at Dartmouth College. Agent Tcl extends the standard Tool Command Language (Tcl) [Ous90], a high-level scripting language that was developed in 1987 and has enjoyed enormous popularity. Agent Tcl addresses four main goals:

- Reduce migration to a single instruction and allow the instruction to appear at arbitrary points, and, once the instruction is called, transparently capture the current state of the agent and transmit this state to the destination machine. The programmer should not have to explicitly collect state information. The system should handle all transmission details, including the possibility of the destination machine being disconnected or having a new network address.

- Provide transparent communication among agents. The communication primitives should be flexible and low-level, but should work identically regardless of whether the agents are on the same or different machines, and should hide all transmission details.

- Provide a simple scripting language as the main agent language, but support multiple languages and transport mechanisms, and allow the straightforward addition of a new language or transport mechanism.

- Provide effective security in the uncertain world of the Internet.

Figure 4 shows the architecture of Agent Tcl. The agent server keeps track of the agents that are running on its machine, accepts and authenticates agents arriving from other hosts and restarts these agents in their own interpreter, and provides inter-agent communication facilities. All other services

are provided by agents. Such services include navigation, network sensing, group communication, fault tolerance, and access control. The agents themselves are separate processes running the appropriate language interpreter, which has the capability to capture the agent's state and send this state to a remote agent server.

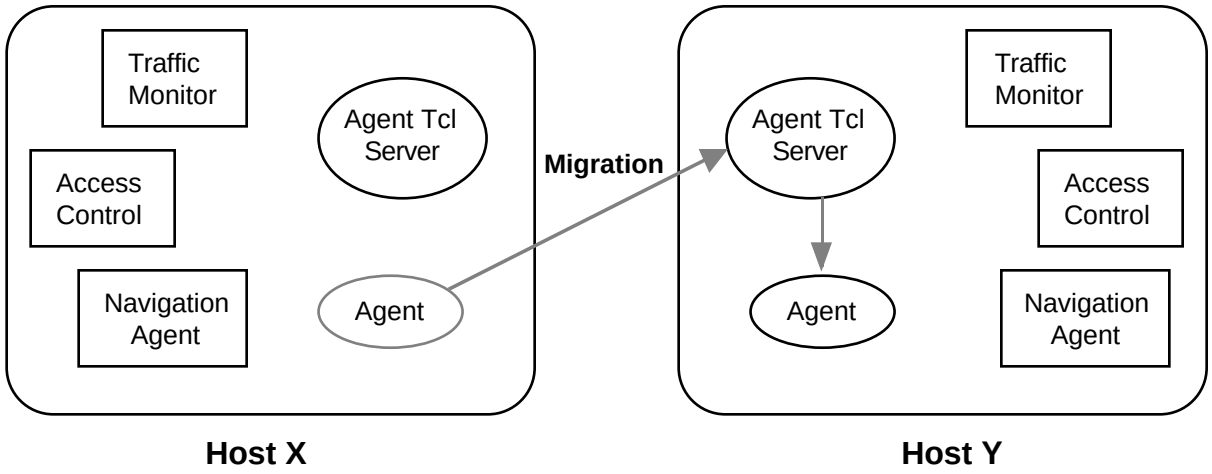


Figure 4 The server-based architecture of Agent Tcl. The agent server coordinates the activities of all local agents and accepts new agents that are arriving from other machines. All other services are provided by specialized agents such as the navigation agent and access control agent

There are various advantages of using Tcl for implementing agents. Tcl is easy to learn and use due to its simplicity and its familiar imperative style. Tcl is interpreted so it is highly portable and easier to make secure, and, in fact, the existing Safe Tcl package already supports the secure execution of untrusted Tcl scripts [Bor]. Tcl can be embedded in other applications, allowing these applications to implement part of their functionality with mobile Tcl agents. Finally, Tcl can be extended with user-defined commands, which makes it easy to tightly integrate agent functionality with the rest of the language.

A set of special commands was added to Tcl to create Agent Tcl. An agent uses these commands to migrate from machine to machine and communicate with other agents. The most important command is `agent_jump`, which migrates an agent from one machine to another. The `agent_jump` command captures the internal state of an agent, encrypts and digitally signs the state image, and sends the state image to the agent server on the destination machine. The destination server restarts the agent from the exact point at which it left the previous machine.

An alternative to `agent_jump` is `mobile_jump`, another Agent Tcl command for migration if the network connection is not constant. `mobile_jump` uses a support system of dock machines

[GKN96] to ensure that agents that want to jump to or from a mobile machine, like a laptop, can do so at the earliest opportunity.

The Agent Tcl system also provides various communication mechanisms (message passing, direction connections, and remote procedure calls (RPC) [Nog96]) for agents to interact efficiently and effectively. Other details about Agent Tcl can be found on the Web[1].

5. Design of DartFlow

DartFlow is composed of four salient components.

- The user interface.
- Transportable process-agents that carry the data and flow control information.
- Agent servers that implement major functionality like information routing, error-handling etc.
- The worklist server to communicate the results from the process-agents to the user interface.

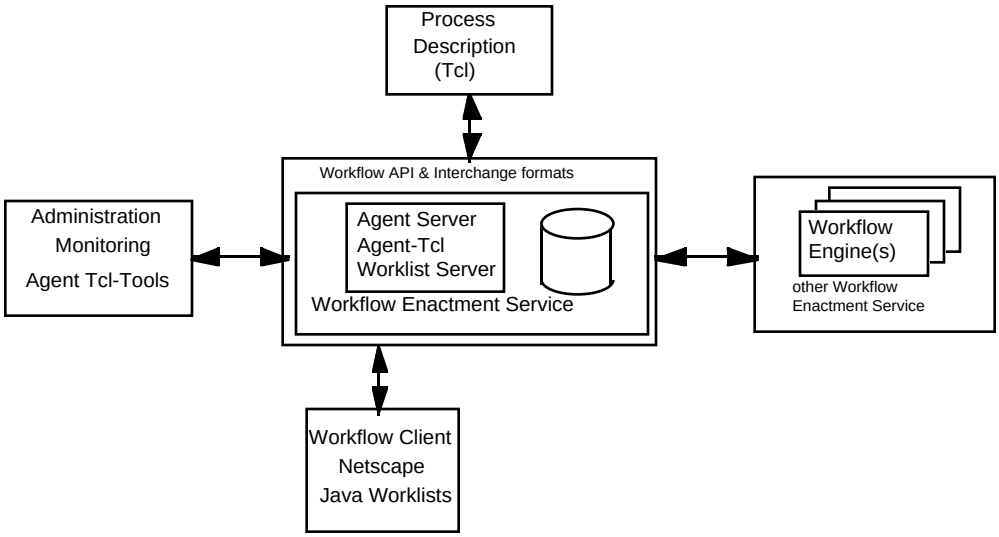


Figure 5 DartFlow System Architecture in the WfMC framework (cf. Figure 1)

Figure 5 illustrates the overall system architecture and implementation of the DartFlow system. This section discusses the four main components as depicted in Figure 5.

[1] `http://www.cs.dartmouth.edu/~agent`.

5.1. User Interface

DartFlow's user interface employs Java-enabled [Sun94] web browsers. Though not strictly required (alternatives are Tk [Ous94] or X11 programming), the web browser based UI is very useful because it provides a platform independent user interface. DartFlow's UI also employs Netscape frames, which are quickly becoming a standard feature of all web browsers. A snapshot of the user interface is shown in Figure 6. The top frame consists of a series of buttons, which link to the appropriate forms. Thus, if users wish to fill in a bank account application form, they click on the appropriate button and a new form appears in the bottom frame of the browser.

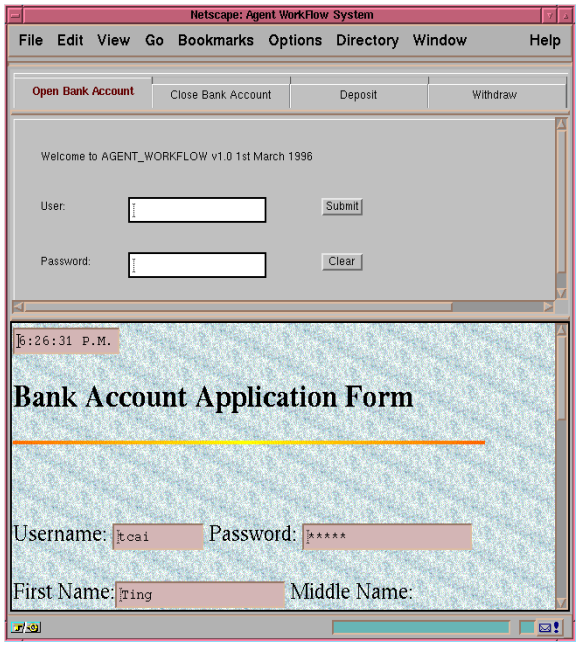


Figure 6 Initial screen of the on-line bank account opening process (cf. section 6)

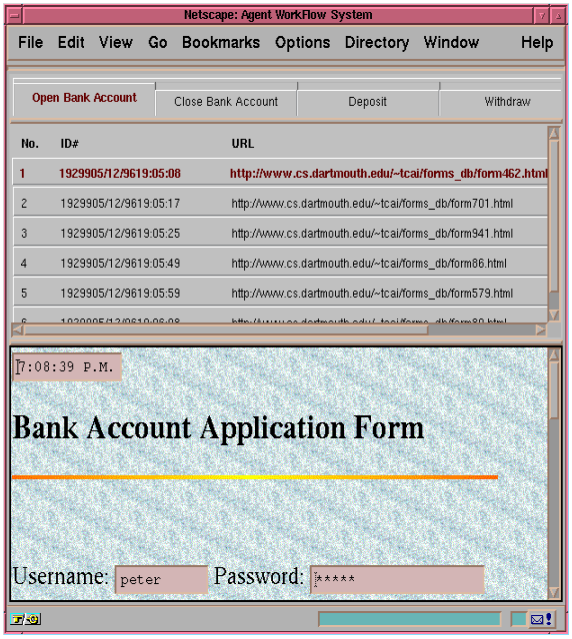


Figure 7 Worklist applet

The middle frame displays the user's worklist (Figure 7). The worklist applet gets information about the items waiting in the worklist from the worklist server (see 5.3). When the applet starts, it brings up a login screen asking for the username and password (Figure 6). This provides the standard password-based security mechanism for DartFlow and ensures that users can only look at their worklist after they have been authenticated. After successful authentication, the applet establishes a socket connection to the worklist server which resides on the same machine as the HTTP Web server. This is a limitation imposed by the security features of Netscape; a Java applet running inside Netscape can only open a connection to the server from which it was loaded. The applet then sends the user name to the worklist server. The worklist server, on receiving the request, sends the contents of the worklist for that user to the applet where it is displayed. At any given point in time, users may

have many concurrent applets accessing their worklist. The worklist server takes care of keeping all of them synchronized and up to date.

Clicking a worklist entry brings up that particular form on the user's web browser. At this point, users can fill out or modify parts of the form and resubmit it.

5.2 Process Agents

The submission of a modified form by any user invokes a CGI script. The script either spawns a new to carry the data or wakes up an old agent if the form was modified and not submitted for the first time. A single transportable agent is responsible for taking care of a process instance from start to finish (hence the name process-agent). A unique ID is attached to each new form (which is also treated as the ID of the process-agent) in the system. These IDs are never reused so as to enable each submitted form to be uniquely identified. The agent parses the contents of the form into its constituent fields and then proceeds to complete its task with the help of Agent Servers (see 5.3) which provide the business logic in DartFlow. The process-agent jumps to the machine where the intended agent server is and then either initiates a meeting or uses the higher level Agent RPC [Nog96] to communicate with the appropriate server. The server, after processing the request, sends the results back to the agent. Tcl allows these results to be Tcl scripts which the agent can evaluate to dynamically modify its behavior or its content. This mechanism provides certain distinct advantages:

- *Adaptation and Specialization:*
  Since each transportable agent contains its own process description, DartFlow has the capability to adapt each instance of a task to its specific needs. Each of the tasks grows and evolves independent of the other tasks in the system allowing DartFlow to deal flexibly with events as they unfold.
- *Extendibility*

  Tcl, being an interpreted language, provides many advantages like enhanced portability that are not available to programs written in standard programming languages like C or C++. Tcl agents can send or receive Tcl procedures and evaluate them. Thus a running agent can overwrite its existing procedures or create new ones on the fly. This is one way in which agents can "learn" things. It also makes the task of writing the agent servers much easier because the servers can send scripts to agents that, when evaluated, make the agents do the right things depending on their individual constitution (variables, declared functions etc.). This script evaluation way of passing information is used by the organization server (see 5.3) to convey activity trajectories to newly created process-agents.

Each step in a process (“task”) might consist of many task steps (“activity”). Thus the process-agent performs arbitrarily complex processing (like accessing databases or devices like printers etc.) to accomplish each task. When the task has been completed and it needs the attention of another human being in the system, the agent jumps to the worklist server (see 5.4). There it saves the data it has been carrying to the disk and sends a message to the worklist server requesting it to add an entry to the worklist of the next recipient (the process-agent knows who the next person in the hierarchy is). The form appears in the Java worklist of the target person and the process repeats from here on. The agent then saves itself to disk in a form that can be restarted when the next user submits the modified form.

5.3 Agent Servers

The agent servers[2] contain the core functionality of DartFlow. They contain and protect vital organization and process specific knowledge, while making relevant parts of it available to the agents as and when necessary. As an agent progresses towards its goal of successfully completing the business process, it talks to various servers that provide it with the information and functionality that it needs to accomplish the individual tasks.

Presently we have implemented two agent servers, that provide the flexible base structure around which DartFlow is built. These are the *organization hierarchy server* and the *tracking server*.

- The company's chain of command is encapsulated inside the **organization server**. The server contains generic process flow information for each type of process as well as information about the structure of the company. A newly created process-agent has no knowledge of the organization except that its first step needs to be to query the organization server. The process-agents thus bootstrap themselves by querying the organization server with their type (the process they are implementing) and their source (the user-ID of the person who generated the form). The organization server returns the task list (ordered list of tasks) to be followed by the process-agent to complete its mission. This task list is computed by mapping the generic process flow information to the particular instance that the agent is implementing. Thus, the organization server implements information routing within the DartFlow system.

  Since the server is a separate entity, it can implement arbitrarily complex models of organization without increasing the complexity of an individual process-agent. A company may decide to modify the organization server at any time (organizations are changing all the time) and the

[2] DartFlow agent servers are agents acting as servers. They are different from the agent server which is part of Agent Tcl architecture.

changes will be apparent as soon as the server has been updated. This design thus provides a flexible organization structure without placing additional complexity and processing load on the rest of the workflow system. There are also various advantages in letting each process-agent carry its complete routing information. While it reduces server interaction and performance bottlenecks substantially, it also allows the process-agent to perform intelligent routing. The agent can take advantage of the inherent parallelism in the process in a way best suited to the present circumstances.

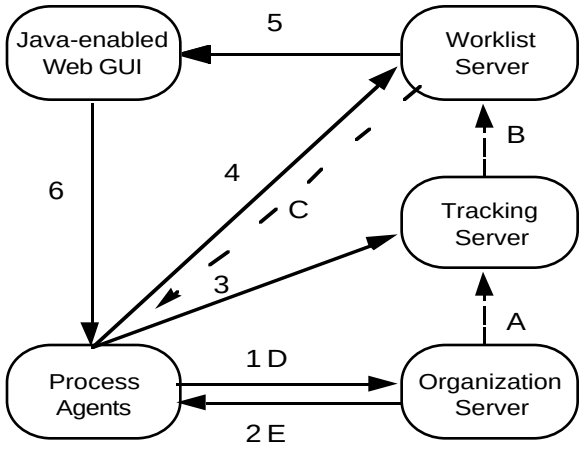
- Allowing process-agents to carry their complete task list has many advantages. However, it increases the response time of the system to changes because the process-agents that have already been launched are self contained and hence to do not interact with agent servers on a regular basis. Thus changes made to the knowledge base of the servers is not immediately reflected in the behavior of process-agents. The **tracking server** solves this problem in DartFlow by coordinating with the **worklist server** (see 5.4). Each process-agent in the system has to contact the worklist server at the end of each task it accomplishes to display the results to the next person in the hierarchy.

  Each newly created agent, after getting its task list from the organization server, goes and registers itself with the tracking server using the 3-tuple <*agent_ID, process_type, task list*>. The tracking server thus keeps track of all alive agents and their sequence of activities. When something changes in one of the agent servers it notifies the tracking server about the affected parts. So if the organization server changes, the tracking server is informed about the affected activity trajectories. The tracking server, using the information it has about all live process-agents, figures out the list of affected agents and sends it to the worklist server. When that process-agent comes to the worklist server to display the updated forms, it is notified to return to the appropriate server (only the organization server presently). Thus the process-agent is informed that the system has been modified and it can get the modified information from the appropriate server.

  Changes take effect at the end of a process-agent's current task. Hence all affected agents will be modified before they begin their next task. This is pretty good response behavior and coupled with the fact that the tracking server generates minimal network traffic (it does not keep track of where an agent is presently) makes the solution even more attractive.

  The last thing a process-agent does before it exits, having completed its task, is to jump to the tracking manager and inform it that it is no longer running. This helps in keeping the information available to the tracking server updated.

Figure 8 gives an overview of the way the various components of DartFlow interact with each other.



ormal Process:

. Process-agent sends the process type and source D.
. Organization server returns the task list.
. Process-agent registers with the tracking server.
. At the end of each task, process-agent informs the orklist server.
. Worklist server updates the display.
. Users submit new data, returns to step 4.

Dynamic Changes:

A. In case of any changes in the organization chart, the organization server informs the tracking server of the affected users and processes.
B. Tracking server informs the worklist server of the affected agents.
C. Next time process-agent contacts the worklist server, the worklist server checks if that agent is affected.
D. Process-agent contacts the organization server.
E. Organization server returns the new work list.

Figure 8 Overview of DartFlow

The agent servers thus form the backbone of DartFlow and provide its flexibility. They make the system very easy to modify and maintain while protecting the domain-specific business logic. Since the system consists of functionally independent agent servers it makes DartFlow scale well while allowing for graceful degradation in case of failures.

We are currently in the process of implementing an error-handling server for DartFlow. The error-handling server will be contacted in case a running process-agent encounters an error condition. The error-handling server may initially try some simple solutions like redirecting the agent to another server or, in case of a problem, alerting the system administrator. This method allows for flexible on the fly error handling, an elusive feature in most of the commercial workflow systems, and relieves the process designer of having to anticipate complex error conditions.

5.4 Worklist Server

The worklist server performs a couple of major functions. It maintains the worklists and updates information on active clients and also informs process-agents that they have been recalled by some agent servers(s) (based on the information sent by the tracking server). The worklist server has been

written in C because of the amount of list management required. Because of Netscape's socket security features for downloaded Java applets, the worklist server has to reside on the same machine as the web server (httpd). Each process-agent, once it successfully completes a task from its task list and requires human attention, sends a message to the worklist server, adding itself to the worklist of the next user.

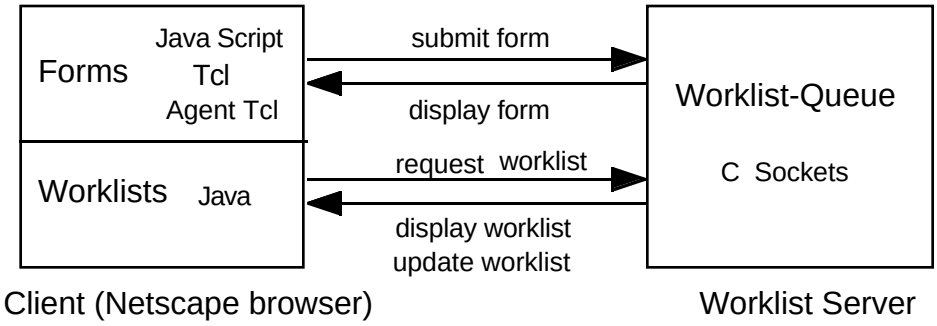


Figure 9. Communication protocol between worklist server and GUI

The Java GUI maintains a socket connection with the worklist server and whenever anything is added or deleted to the worklist of the user, the Java applet is immediately notified and updates the client (Figure 9). A user logged in from multiple places simultaneously will see the same worklist.

6. A Sample "Real World" Process: Opening a Bank Account

DartFlow has been successfully used for implementing some simple but real-life business applications. We present here a hypothetical bank account-opening process "on the Web".

Figure 10 displays the outline of our bank account opening process. The arrows indicate flow of information. Customers log into the DartFlow system as new-user and fill out the HTML form presented to them by the browser. The form asks for general information like name, social-security number, address, yearly income, initial deposit amount, etc.

When users have filled out all the details, they press the "Submit Form" button and the associated CGI script generates a process-agent to take care of the new application. The process-agent, first gets its task list from the organization server and then registers itself with the tracking server. Using the information provided by the organization server the process-agent goes to a Credit-History database. This database may be in-house or provided by some other independent credit-rating agency. There the process-agent meets with the database agent (which guards access to the actual database and prevents misuse) and forwards its query to it. The database agent performs the query and returns the result to the process agent. The result contains either a credit-history report or an error condition like

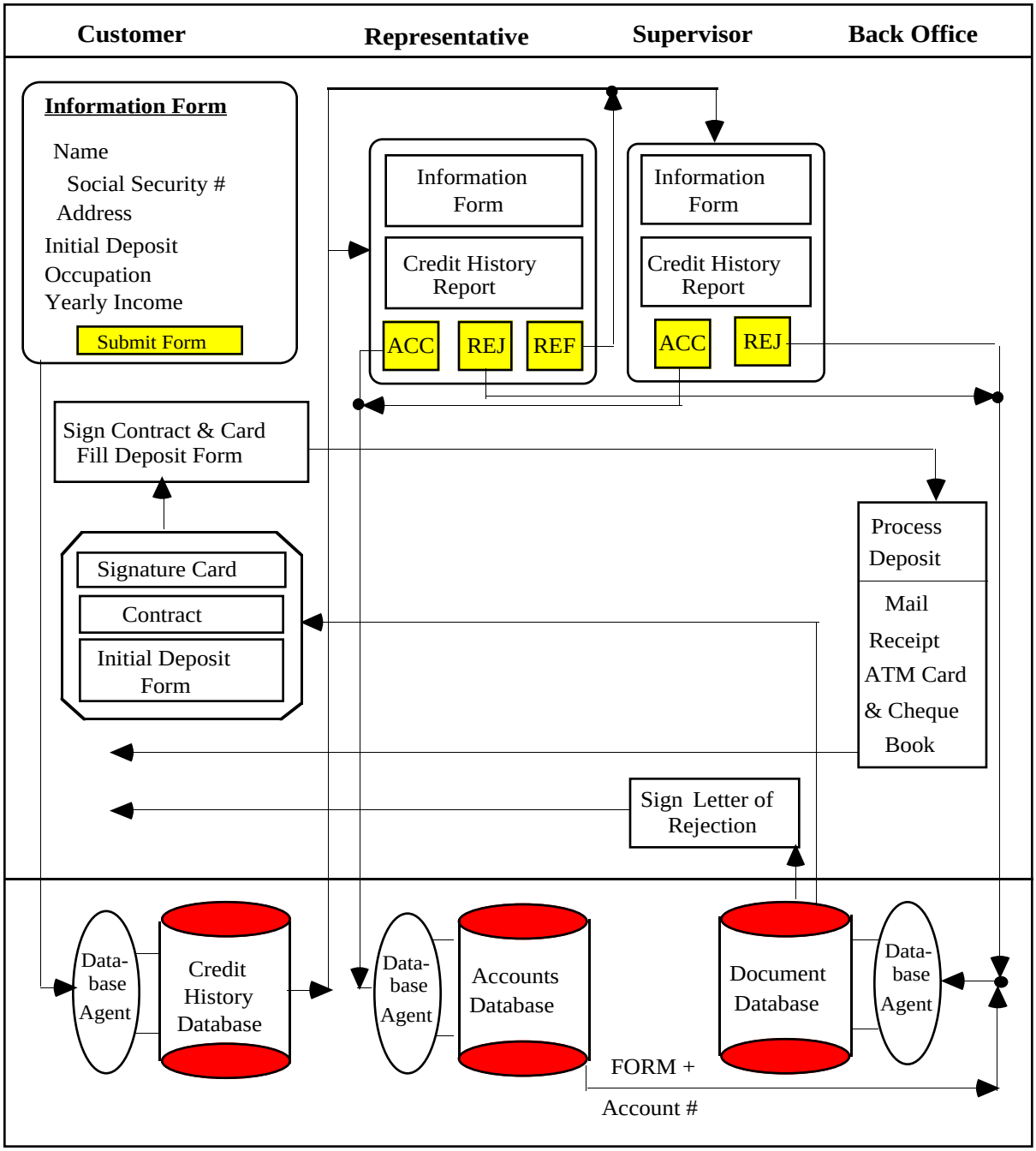


Figure 10 Sample opening of an account over the Web

- The name and social-security number do not match.
- The database has no record for this social-security number.

The process-agent,, at this point, may decide to query other databases. Eventually, when enough credit-history information is available, the process-agent decides to jump to a customer representative or a manager depending on the amount of initial deposit and the type of deposit. This is an internal check mechanism built to prevent the bank from being involved in money laundering. In our case, if

the deposit is greater than $2000, the form is routed to a manager, otherwise to a customer representative. Depending on the credit history report the customer representative may decide to accept, reject, or refer the application to the manager. The manager decides on these referred cases as well as on high-deposit (> $2000) cases.

If the application is rejected, the process-agent meets with a document database agent and gets the template of a standard rejection letter. It then fills in the appropriate information like the name, address, and the address of the credit-history provider. The letter is printed, probably on the printer in the manager's office, and thus appears on the manager's desk. At this point the DartFlow process-agent has completed its task and exits. The manager signs the letter (as a good PR exercise and to provide the user with a person to whom to address complaints) which is then mailed to the user.

If the application is accepted, the process-agent meets with the accounts database agent, sets up the account information, and gets an account number for the new account. It then retrieves the welcome letter, appropriate contract forms, the signature card, and the initial-deposit form from the document database. The process-agent fills in the appropriate information and either prints the documents in the back office to be mailed to the user, or allows the user to print the documents on the local printer. The user signs the contract and signature card, fills in the initial deposit form and mails everything to the bank. The documents are received and handled by the back-office. The deposit is made and the user is mailed a receipt, an ATM card and a check book.

DartFlow offers major advantages over traditional workflow systems even when implementing a simple process like the one described here:

- There is a thread of control attached to each submitted application. Thus, a process-agent can take individual actions depending on how it progresses. For an applicant with excellent credit history the process may be shortened while additional databases might be consulted for an applicant who either doesn't have an extensive credit history or is a borderline case.
- While an agent carries its routing information with itself, the general “role to specific user” mapping is done for each case individually. So, if on a particular day a customer representative is not available, the system will directly reroute new applications to other customer representatives’ worklists.
- Agents can deal intelligently with unexpected error conditions that occur during the process. If, for example, a particular social-security number does not exist in the regular database, the agent can decide to consult alternate databases. If it fails there, too, it may decide to log an error report and ask for human intervention.

7. Summary and Future Work

In this paper we have discussed the design and implementation of DartFlow, a workflow management system, using two emerging technologies: the world wide web and transportable agents. The emphasis was on ameliorating some of the nagging problems like lack of flexibility, adaptation, specialization, and intelligent error handling that plague commercial workflow systems. We still have a long way to go for our stateless agent-based workflow system to reach the level of reliability and performance required for real business processes of real companies. Nevertheless, we are convinced that our approach offers new and extremely promising solutions to general workflow problems that are hard to solve with conventional systems.

We plan to implement a real time on-line connection to a large repository of generalized business processes, the MIT process handbook [Mal93]. The idea is to use the process handbook as a rule base of organizational knowledge for the agent that can be consulted in case unanticipated situations arise.

Another concept that can be nicely supported by agent based workflow is to enable agents to learn in order to improve process quality. To improve the quality of the workflow process, the autonomous agent could be made capable of learning during its lifetime by augmenting or modifying its own functionality as well as improving a persistent knowledge base of organizational knowledge.

On a more practical and implementation oriented level we are exploring options to make the DartFlow system more scalable and more easily distributable by getting rid of a dedicated socket for worklist management.

Acknowledgments

Thanks to Bob Gray for helping us with Agent Tcl and letting us borrow the section on Agent Tcl design from his original paper [GKN96]. We are also thankful to Professors George Cybenko, Fillia Makedon, and David Kotz and Bob Gray for their help in reviewing this paper.

Bibliography

[Alo96] G. Alonso, H. Schek. *Database Technology in Workflow Environments.* Informatik, 2/1996, February 1996.

[Ber92] T. Berners-Lee, R. Cailliau, J. Groff, B. Pollermann. *World-Wide Web: The Information Universe.* CERN, Geneva, Swizerland. 1992.

[Bor] N.S. Borenstein, M. Rose. *Safe Tcl.* Available at `ftp://ftp.fv.com/pub/code/other/safe-tcl.tar.Z`.

[Bus94] C.J. Bussler. *Policy Resolution in Workflow Management Systems*. Digital Technical Journal. vol 6, no 4, fall 1994.

[Dec95] Digital Equipment. LinkWorks White Paper. 7/9/95. (www.digital.com/info/linkworks/)

[GKN96] R. Gray, D. Kotz, S. Nog, D. Rus, G. Cybenko. *Mobile Agents for Mobile Computing*. Technical Report: PCS-TR96-285, Department of Computer Science, Dartmouth College, 1996.

[Gra95] R. Gray. *Agent Tcl: A Transportable Agent System.* Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM), James Mayfield and Tim Finnin, editors, December 1995.

[Gra96] R. Gray, D. Rus, D. Kotz. *Transportable Information Agents*. Technical Report: PCS-TR96-278, Department of Computer Science, Dartmouth College, 1996.

[Ham93] M. Hammer, J. Champy. *Reengineering the Corporation*. HarperCollins, New York. 1993.

[Ley95] F. Leymann, D. Roller, E. Vogt. *White Paper: Workflow Management.* IBM Software Solutions Division, German Software Development Laboratory, Boeblingen. March 8, 1995.

[Mal93] T. Malone, K. Crowston, J. Lee, B. Pentland. *Tools for inventing organizations: Toward a handbook of organizational processes*. Sloan School Technical Report WP # 3562-93.

[Nog96] S. Nog, S. Chawla, D. Kotz. *An RPC mechanism for Transportable Agents.* Technical Report: PCS-TR96-280, Department of Computer Science, Dartmouth College, 1996.

[Ous90] J. Ousterhout. *Tcl: An Embeddable Command Language* . Winter 1990 USENIX Conference Proceedings. 1990.

[Ous94] J. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Professional Computing Series, Addison-Wesley, 1994.

[Sun94] Sun Microsystems. *The Java Language: A White Paper*. Sun Microsystems,1994.

[The95] L. The. *Workflow Tackles the Productivity Paradox.* Datamation. August 15, 1995.

[WfM95] Workflow Management Coalition, International Orgnization for the Development and Promotion of Workflow Standards. *Glossary*. Workflow Management Coalition, Avenue Marcel Thiry 204, 1200 Brussels, Belgium. 1995.