# RainMan: A Workflow System for the Internet

Santanu Paul, Edwin Park, and Jarir Chaar

*IBM T. J. Watson Research Center*
*P.O. Box 704*
*Yorktown Heights, NY 10598*

## Abstract

*As individuals and enterprises get interconnected via global networks, workflows that scale beyond traditional organizational boundaries and execute seamlessly across these networks will become relevant. We address the problem of designing a scalable workflow infrastructure for the Internet that supports both flexibility in workflow participation and interoperability between heterogeneous workflow system components.*

***RainMan*** *is a distributed workflow system developed in Java that lives naturally on the Internet. RainMan is a loosely-coupled collection of independent services that cooperate with each other rather than a monolithic system. Some of the useful features of RainMan are browser-based workflow specification, participation, and management, and dynamic workflow modification. The RainMan system is based on* ***RainMaker****, our generic workflow framework that defines a core set of well-defined interfaces for workflow components.*

## 1. Introduction

Workflow systems are essential to organizations that need to automate their business processes [Silver95]. The attraction of these systems is that they help organizations specify, execute, and monitor their business processes in an efficient manner over enterprise-level networks [FlMa, VisFlo, InConc, ActWork]. Workflow systems provide improved throughput and tracking of processes, and better utilization of organizational resources.

As individuals and organizations become interconnected via global networks such as the Internet, they will attempt to team up in various ways to share work and processes *across* traditional organizational boundaries. An Internet infrastructure that enables such interactions as workflows would be of great value; unfortunately, traditional workflow systems designed for centralized workflow execution do not lend themselves well to these new workflow applications. Internet-wide workflows require an infrastructure that supports decentralized workflow execution, workflow system interoperability, dynamic workflow modification, and low-cost workflow participation. The proprietary and monolithic design of current workflow systems makes it very difficult to address these requirements.

The *Workflow Management Coalition* (WfMC) has proposed a reference architecture and defined interfaces for vendors of traditional workflow systems to interoperate [WfMC]. The WfMC standard is a useful step in the direction of interoperability, however, its main shortcoming is that it promotes a monolithic workflow system architecture that is not flexible or scalable enough to address the needs of Internet-wide workflow applications which must necessarily be loosely-coupled.

We designed *RainMan* as a distributed workflow system for the Internet. The system comprises a collection of independent and lightweight services on the network. Our Java implementation uses open standards and Web browser-based user interfaces. We are using RainMan to experiment with a range of interesting features such as *dynamic workflow modification* which allows a workflow graph to be changed during execution, *disconnected participation* which allows participants, designers and administrators on the Internet to be infrequently connected to the network, and *downloadable workflow execution* which allows workflow subprocesses to be downloaded across the Internet on demand.

RainMan is based on *RainMaker*, our generic workflow framework that defines a core set of abstract interfaces for workflow components. The main purpose of RainMaker is to facilitate the design and implementation of flexible, interoperable, and scalable workflow system components.

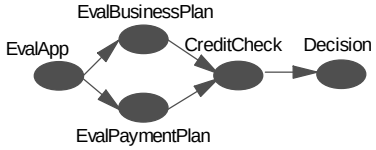


Figure 1: A Business Loan Approval Workflow

## 2. Workflow Systems Background

The traditional use of workflow systems has been in the automation of highly structured, process-based applications that are common to banks and insurance companies. Figure 1 shows a typical (simplified) *BusinessLoanApproval* workflow in a bank. Workflow systems provide extensive support for the design and automation of such applications in an efficient manner. Some of the important features they provide are:

**1. Specification Tools**: These are used to specify processes using high-level specification languages. There is no commonly-accepted basis for specification languages, each workflow system imposes its own language which can be based on directed acyclic graphs, rules, state machines, Petri-nets, or other any formalism. In general, it is common practice to program workflows visually as a directed graph where the nodes represent activities and the arcs represent control and data flows between activities.

**2. Execution environments**: A workflow system provides an execution environment for workflow instances; this involves interpreting the process by stepping through the activities and assigning them to appropriate organizational resources such as humans, applications, and other workflow systems.

**3. Audit Logs and Tools**: These are necessary to monitor and track the progress of workflow instances through the workflow system.

**4. Worklist Management**: Management of *worklists*; worklists are persistent storage locations where human participants receive work assigned to them.

In effect, traditional workflow systems are sophisticated process specification and execution environments that allow organizations to run process-based applications efficiently, within the context of a closed environment over which the workflow system has significant authority.

In response to a growing number of proprietary workflow systems, the Workflow Management Coalition has defined a set of interfaces to enable workflow system interoperability. These interfaces are tied to a workflow system architecture known as the WfMC Reference Model (Figure 2) which defines interfaces between workflow servers and other components such as process definition or specification tools, workflow client applications, invoked applications, administrative tools, and other workflow servers. The specific details of these interfaces are described in [WfMC].

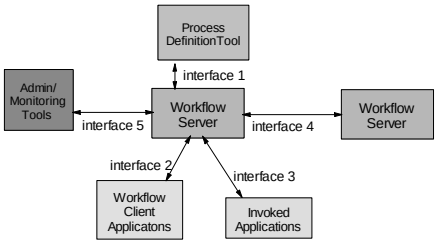


Figure 2: WfMC Reference Model

The architecture defined by the WfMC standard is not well suited for workflow execution on the Internet. The main problem is that the workflow server is monolithic. It is responsible for process execution, auditing, management of the organizational directory, and distribution of activities to appropriate participants (performing role resolution as necessary).

Most importantly, the server is responsible for hosting and managing worklists on behalf of all participants. This is problematic; since worklists are hidden within the workflow server (not externally addressable), activities can be sent only to worklists that reside in the same workflow server. Clients participating in multiple workflows on heterogeneous servers must have a worklist on each server and must manage each of the worklists (Figure 3).

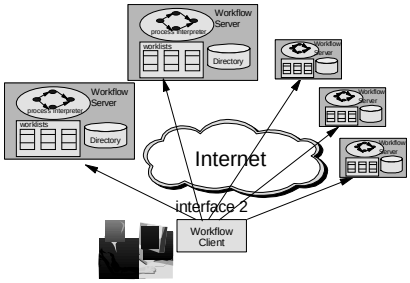


Figure 3: The Pull Model

In addition, the workflow client application must manage multiple connections to each of these workflow servers since the architecture assumes a continuously connected workflow client. This is not a scalable or flexible solution, especially if one is to participate in a large number of workflows on the Internet, or use thin clients or PDAs, or operate in a disconnected mode. An analysis of the WfMC architecture reveals additional problems for decentralized workflow execution; interested readers may examine them elsewhere [Paula97, Schu96].

## 3. New Workflows on the Internet

In RainMan, we explore the potential of the Internet to enable decentralized workflow execution via interoperable workflow components that reside across this global infrastructure. We hope to enable new kinds of workflows involving dispersed individuals, multiple organizations, scattered network resources, and heterogeneous workflow systems. A few motivating examples of workflows that should be possible over the Internet are presented here.

Consider a virtual team of IT consultants from different organizations, scattered across the globe, working on a project that is coordinated via workflow. The consultants may be mobile and intermittently connected to the network. Irrespective of location, the project leader must monitor the workflow and work assigned to consultants must appear on their worklists. Each consultant may participate in many other workflows at the same time.

In such ***decentralized workflows***, participants must receive work from multiple workflows using heterogeneous clients ranging from desktops to laptops to PDAs (Figure 4). The work distribution model needs to be different from that proposed by WfMC. Instead of participants connecting to workflow servers explicitly to pull their work, the workflow infrastructure must automatically route work to them over a global network.

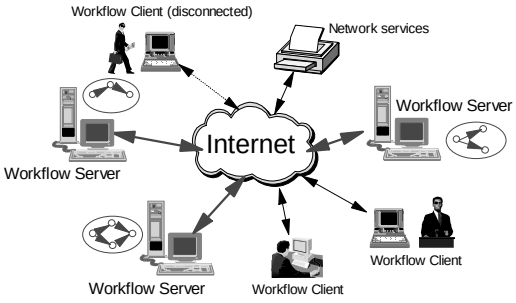


Figure 4: Decentralized Workflow Execution

Next, consider the *BusinessLoanApproval* workflow in Figure 1 running on a bank's workflow server. The *CreditCheck* step may itself be a nested subprocess that is delegated to a CreditEvaluation firm with a workflow server supplied by a different vendor. During execution, the bank's server would notify the firm's server to start an instance of its local *CheckCreditRatings* workflow. On completion of the subprocess, the bank's server would resume its suspended workflow (Figure 5). With global connectivity, these ***peer-to-peer workflows*** between heterogeneous servers are possible. In fact, Interface 4 of the WfMC standard is designed to enable such workflow interactions. However, the drawback of the Interface 4 approach is the high cost of setting up such interactions, and significant pre-agreement required between participating servers.

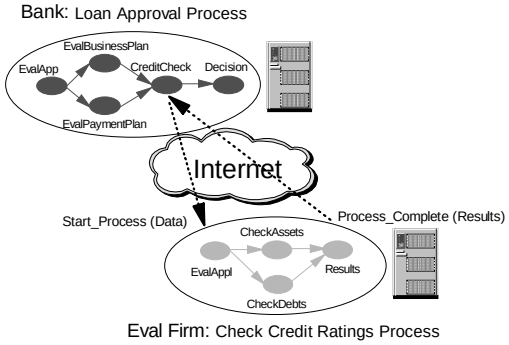


Figure 5: Peer-to-Peer Workflow Execution

Finally, as electronic commerce applications become available on the Internet, it is conceivable that workflows may be downloaded for just-in-time execution. ***Downloadable workflows*** make sense especially in cases where pre-installing and maintaining workflows is not cost effective. For example, consider an education brokerage service on the Internet [Hama96] that specializes in locating custom education services (Figure 6).

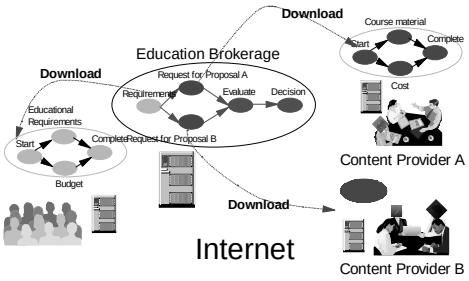


Figure 6: Downloadable Workflow Execution

In a plausible scenario, the *Brokerage workflow* downloads a *RequirementGathering* subworkflow to the client organization and requests its execution. Next, the *Brokerage workflow* downloads the requirements gathered to content providers along with a *RequestForProposal* subworkflow that the latter must execute to create the proposal. At the end, the brokerage compares all the proposals received and notifies the client.

To enable workflows such as these on the Internet, significant issues related to security and organizational privacy must be addressed. Such services, based on state-of-art distributed systems security considerations,

will have to coexist with the workflow infrastructure.

# 4. RainMaker Workflow Framework

RainMaker is a generic workflow framework we have developed to build interoperable workflow components. The details of the framework are available elsewhere [Paulb97]; only a brief outline is presented in this section.

RainMaker identifies four important abstractions within the workflow domain. Workflow instances are considered **Sources**, or service requesters. Sources generate **Activities**, or service requests that are delegated out. Instances of humans, applications, organizations, and other entities that handle these delegated requests are considered **Performers**, or service providers. Performers manage units of work called **Tasks**, which implement the delegated requests issued by Sources. It is an important characteristic of the workflow domain that Tasks can be long-running. A central feature of RainMaker is that a Task may be a workflow instance that recursively acts as a Source; this provides natural support for delegation of subprocesses.

The core RainMaker interfaces that help support this execution model are*:*

- *PerformerAgent:* An abstract interface that is implemented by Performers on the network. The interface provides mechanisms for delegating, controlling, and querying Tasks on the Performer.
- *SourceAgent:* An abstract interface implemented by Sources on the network. It provides a callback mechanism for Performers to return the results of Tasks to Sources.

For the remainder of the paper, the italicized *SourceAgent* and *PerformerAgent* refer to the RainMaker abstract interfaces. The terms Source and Performer are used generically to refer to entities (or objects) that implement the RainMaker *SourceAgent* and *PerformerAgent* interfaces respectively.

The *PerformerAgent* and *SourceAgent* interfaces describe how proprietary Sources and proprietary Performers can interact with each other (Tables 1 and 2). In essence, the *PerformerAgent* interface conceals the internals of how a Performer or service provider actually performs Tasks in response to the Task requests. Symmetrically, the *SourceAgent* interface is a callback interface implemented to conceal the internal details of how the Source actually generates Activities, issues Task requests, and handles responses from Performers. The interfaces also describe the control mechanisms by which Tasks can be suspended, resumed, and aborted; and query mechanisms by which their status can be tracked. The interaction between Sources and Performers is shown in Figure 7.

| *PerformerAgent interface* |
|---|
| `List(TaskDefinition)::listTaskDefinitions()` |
| `TaskID::createTask(SourceAgent source, ActivityID activityid, TaskRequest taskreq)` |
| `AbortID::abortTask(TaskID taskid)` |
| `SuspendID::suspendTask(TaskID taskid)` |
| `ResumeID::resumeTask(TaskID taskid)` |
| `TaskStatus::queryTask(TaskID taskid)` |
| `List(TaskID)::listTasks()` |

Table 1: *PerformerAgent* interface

| *SourceAgent interface* |
|---|
| `CompletedID::completedTask( PerformerAgent performer, ActivityID activityid, TaskResponse taskresp)` |
| `RefusedID::refusedTask( PerformerAgent performer, ActivityID activityid)` |
| `ForwardedID::forwardedTask( PerformerAgent performer, ActivityID activityid, PerformerAgent newperformer, TaskID taskid)` |
| `Boolean::seekPermissionToStartTask( PerformerAgent performer, ActivityID activityid)` |

Table 2: *SourceAgent* interface

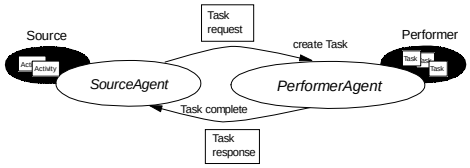


Figure 7: Source and Performer Interaction

In addition to the core interfaces, RainMaker also defines the *Worklist* interface (Table 3). Worklists are an important workflow metaphor similar to electronic mail boxes; they provide a mechanism for human participants to access work assigned to them by workflow systems. In the case of RainMaker, Performer entities that represent human participants on the network can implement the *Worklist* interface and allow participants to view and access the Task Requests sent to them by various Sources.

| *Worklist interface* |
|---|
| `List(WorkItemDescriptor)::getWorklistIndex()` |
| `WorkItemRequest::getWorkItem(WorkItemID wid)` |
| `void::completedWorkItem(`<br>`WorkItemID wid,`<br>`WorkItemResponse wresp)` |
| `void::refusedWorkItem(WorkItemID wid)` |
| `void::forwardWorkItem(`<br>`PerformerAgent newperformer,`<br>`TaskID taskid)` |

Table 3: *Worklist* interface

The strength of the RainMaker framework lies in its generalized *push model* of Task distribution that allows Tasks to be delegated to Performers *independent* of their implementation. This aids the interoperability of heterogeneous workflow systems and components. Implementations of the *PerformerAgent* interface can represent arbitrary service providers: humans, applications, roles, organizational units, and workflow systems. Implementations of the *SourceAgent* interface can embody heterogeneous workflow applications described as rules, as control/data flow graphs, and even non-workflow applications such as collaborations or human Sources.

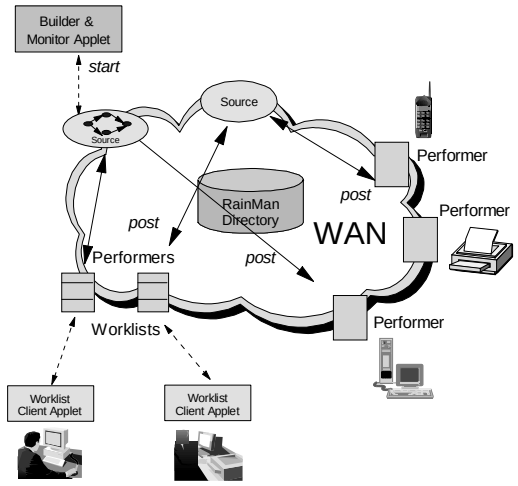


Figure 8: RainMan System Design

# 5. The RainMan System

The RainMan system (see Figure 8) is a distributed workflow system prototype written in Java using RainMaker interfaces. It consists of a loosely- coupled collection of distributed lightweight services required to deliver workflow functionality to Internet users. It currently consists of approximately 60 Java classes developed with Java JDK 1.1.4 and it uses Java's Remote Method Invocation (RMI) for transport between distributed components. The transport has been designed to be replaceable; it can be reimplemented on top of any messaging system. RainMan runs on a TCP/IP token ring network of Wintel machines.

## 5.1 RainMan User Interface Components

The user interface components of RainMan allow workflow users to interact with the runtime environment. The currently available ones are the Builder, the Worklist Client, and the Administrator, all implemented as Java applets. This makes any Java-compliant Web browser a usable client for all RainMan user interfaces, and has the additional benefit of making RainMan user interface components portable and hardware-independent. The Builder and Worklist Client are described in this section.

### 5.1.1 RainMan Builder Applet

The RainMan Builder (see Figure 9) in its current incarnation is a single Java applet that combines three functions. It is an interactive graphical environment for specifying workflows as directed, acyclic graphs. Performers are assigned from the RainMan directory service view available within the Builder. The service-specific aspects of an Activity are handled using the JavaBeans component object model. Workflow graphs can be saved to and loaded from a workflow specification repository.

The Builder also acts as a workflow graph interpreter (Source) and implements the *SourceAgent* interface. For an executing workflow, the Builder posts Task requests to specified Performers. On receiving notification of results, it inspects the workflow graph and posts the next set of Task requests. Since the Builder dynamically interprets the workflow graph, it is possible to change or refine the 'downstream' specification of the workflow graph even after the workflow has been started. This is a useful feature that enables the definition and execution of *ad hoc* processes, and distinguishes RainMan from traditional workflow systems. It allows for dynamic updates to the workflow graph to deal with emerging scenarios.

This is particularly relevant to decentralized workflow execution on the Internet, because we expect Performers to have significant autonomy over their internal domains and hence be capable of raising exceptions in response to Task requests from Sources. In addition, Performers can change their capabilities, as well as join or leave the network at will. Workflow systems that cannot be dynamically modified can be very limiting in such situations.

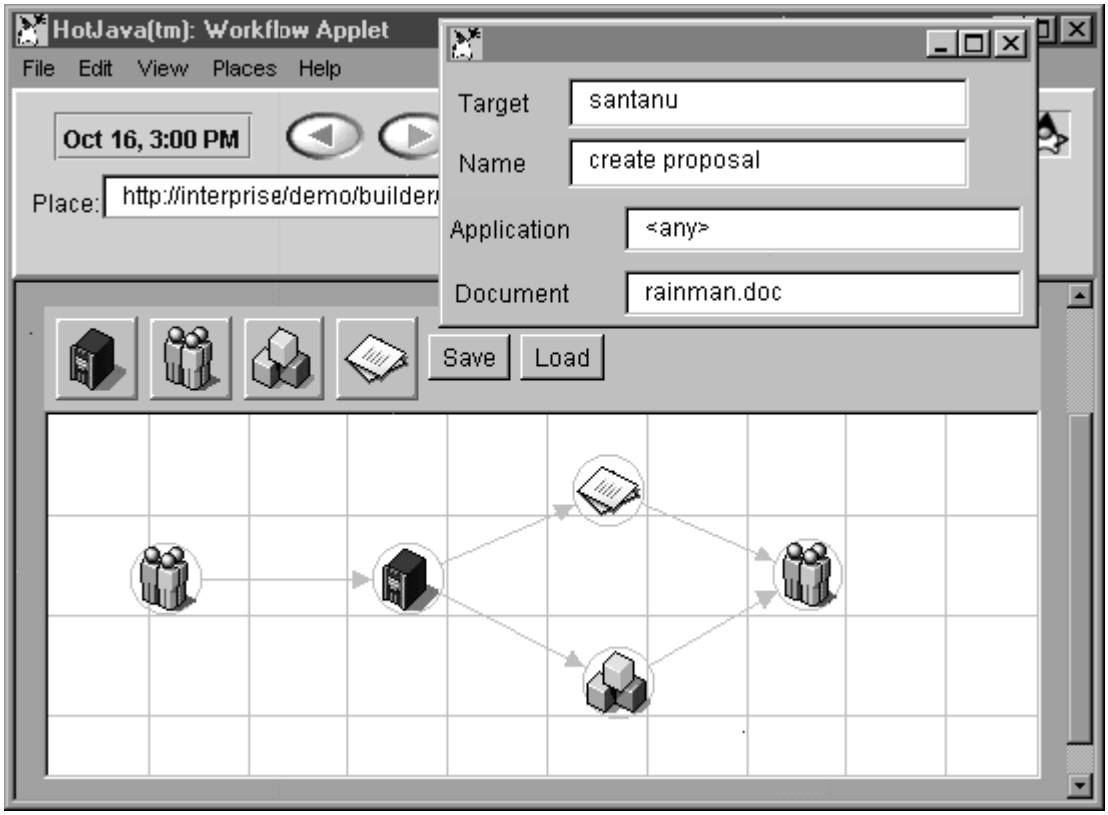


Figure 9: Builder Applet

Support for groups and roles is an important part of workflow execution. Workflow systems are used to improve process throughput; one of the ways to achieve that objective is to assign an Activity to a role at build time. A role represents a group of 'fungible' Performers (i.e. 'designer', 'tester', 'manager', etc.); the binding of an Activity to a specific Performer is postponed until execution. In RainMan, we are experimenting with the use of special-purpose Performers to manage roles; section 5.2.3 covers this topic in some more detail.

The choice of a good workflow specification language remains a matter of considerable debate within the workflow community. RainMan uses a vanilla workflow specification language based on directed, acyclic graphs. It is important to realize that RainMan offers an excellent infrastructure for deploying a wide range of specification tools based on heterogeneous languages, since the details (and potential idiosyncrasies) of a specification language are completely encapsulated within a Source implementation and not exposed either to the Performers or to the RainMan infrastructure. The clean separation of responsibilities of workflow routing and Task execution between Sources and Performers respectively makes RainMan a suitable infrastructure for plug-and-play operation with heterogeneous Sources and Performers.

Finally, the Builder also helps users monitor the state of a workflow execution. It provides visual cues to the owner or administrator about the global state of a workflow in execution at a coarse level of granularity.

Even though the three functions of Builder, Source, and Monitor are currently physically part of the same Java applet, they remain distinct logical entities in our workflow architecture. The next version of our prototype will separate some of these functions, allowing for features such as disconnected and remote building and monitoring.

### 5.1.2 RainMan Worklist Client Applet

The RainMan Worklist Client is a Java applet that allows users to access their Worklists from within a Web browser (see Figure 10). A Worklist is a remote Java object on the network that implements the *PerformerAgent* and *Worklist* interfaces and acts as the Performer for a human participant. The participant can use the Worklist Client to view the contents of the remote Worklist, and selectively download specific Task requests and perform them.

Appropriate Task Handlers are automatically launched to allow the human to interact with Task requests. Each participant or Performer has a set of capabilities in RainMan. For example, for humans capable of document processing, the Worklist Client currently handles incoming *Document_Processing* Task requests via a specialized Task Handler that can locate a referenced document from a network data store and launch the necessary local application (word processor) needed by the human to interact with the document. On completion, the Task Handler returns the response to its Worklist, which in turn returns it to the requesting Source.

Since each Task request received from the Worklist is handled locally on the client machine, disconnected operation is handled easily. The client applet needs to connect to its remote Worklist only for the purposes of receiving and returning Activities. However, a connection to the network may still be needed if the Task needs to reach certain data elements on the network in the course of its execution, or if the applications needed to perform it are not locally available.

The RainMan Worklist Client Applet is significant because it offers a valuable proof-of-concept that demonstrates how human performers can receive work from heterogeneous workflow sources using a single, unified user interface. Current workflow systems usually come with proprietary client applications that provide worklist access by pulling work from their proprietary servers. In contrast, RainMan requires that the Worklist Client pull work only from its designated Worklist on the network, to which Task requests are pushed from various workflow backends. The RainMan approach imposes a much lower burden on the Worklist Client since it no longer has to handle multiple connections, one with each workflow server it is receiving work from. In an interoperable workflow world, an equivalent of the RainMan Worklist Client Applet could replace proprietary worklist clients.

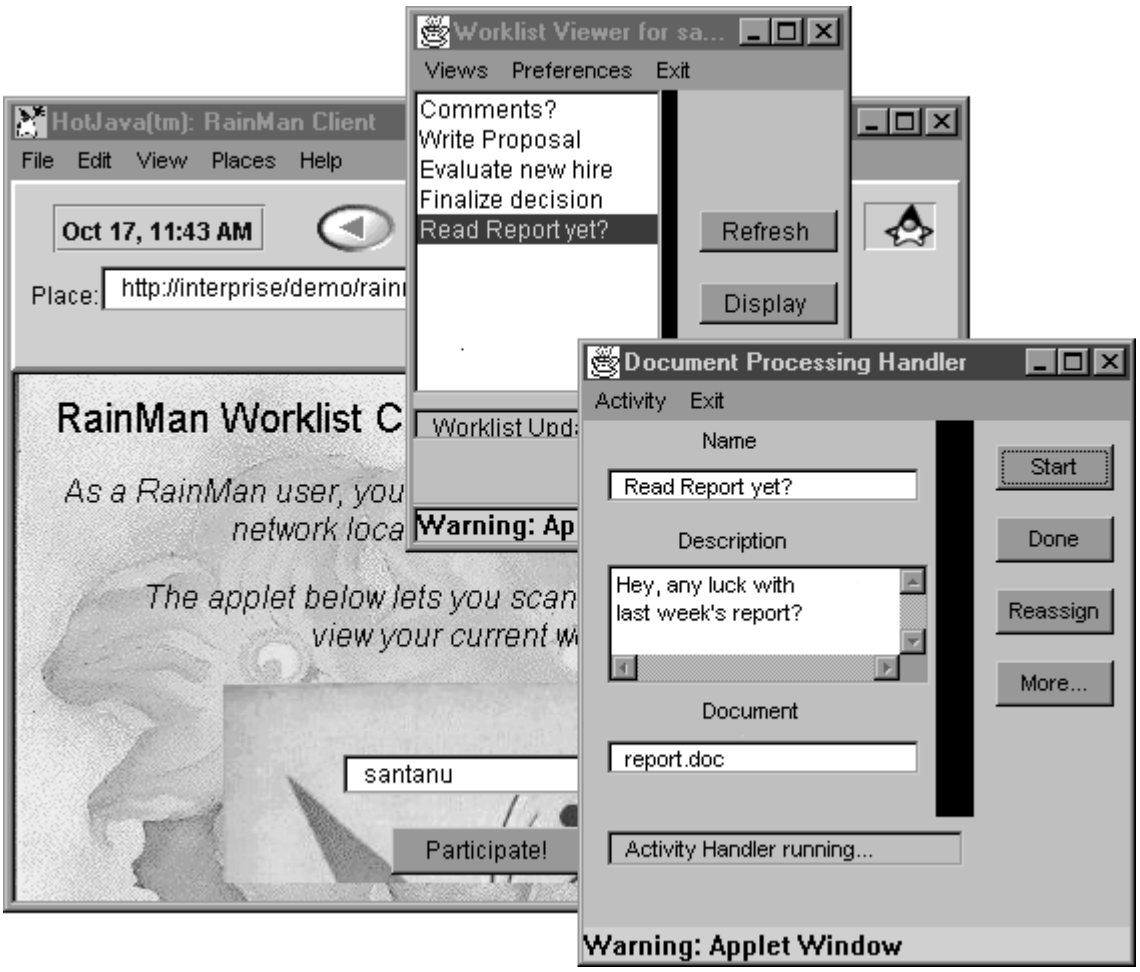


Figure 10: Worklist Client Applet

Task priorities and deadlines, while not yet implemented in RainMan, can be supported. Such constraints, *based on convention and agreements* between Sources and Performers, can be specified as part of the Task request message sent to a Performer. A Performer unable to meet these constraints may refuse to service the request. Similarly, a Source may use the deadline information associated with an Activity to time out on a Performer that does not respond by the deadline. However, attempts to generalize the behavior of arbitrary Sources and Performers with respect to priorities and deadlines leads to the broader question of negotiation protocols between Sources and Performers. Such protocols, while immensely useful, have not been studied in the context of the RainMaker framework.

## 5.2 RainMan Runtime Environment

The RainMan runtime environment provides a collection of distributed services that run on the Internet infrastructure. The user components described in the previous section allow workflow users - workflow designers, participants, and administrators - to interact with these distributed services on the network. The services currently available are the Directory Service, the Worklist Service, and a variety of Performers.

### 5.2.1 RainMan Directory Service

The RainMan directory service plays the key role of a trading service in the RainMan system. It contains information about Performers on the network and their capabilities. When a Source needs to issue a Task request to a Performer, it first performs a lookup operation on the directory service to locate the appropriate Performer on the network. The directory service interface also provides methods to register and unregister Performers. Using the Administrator applet, a RainMan administrator can manage the directory service and its contents. Mobile Performers use the directory service to locate their Worklists as well, thus enabling true location-independent workflow participation. For expediency, the RainMan directory service is currently prototyped as a custom Java application. We are planning an LDAP-based [Yeong95] directory service for the next version of the RainMan prototype.

For widespread workflow deployment on the Internet, it is necessary for the RainMan directory service to be distributed. The design of distributed directory services such as DNS [Mock87] and X.500 [CCITT] can be used as a guideline for RainMan directory design. RainMan directories in individual domains (a domain could be an organization, a geographical location, a collection of smaller domains, etc.) can be hierarchically arranged so that Sources can find Performers across domains. Performers within a domain would always be registered with their local RainMan directory. Sources would always lookup their local RainMan directories for Performers; the local directories could in turn access other RainMan directories by traversing the directory hierarchy to locate matching Performers. A related example is that of the Corba Trading Service [OMG97]; multiple Corba Traders can be explicitly linked into a network for transparent navigation.

### 5.2.2 RainMan Worklist Service

In workflow systems, worklists are inboxes associated with humans. In RainMan, a Worklist is a Java object that implements the RainMaker *Worklist* and *PerformerAgent* interfaces. Therefore, a RainMan Worklist is a Performer that is owned by and represents a human on the network (this is not a limiting assumption - Worklists can easily represent applications as well). Worklists offer persistent storage of Task requests posted to humans from Sources.

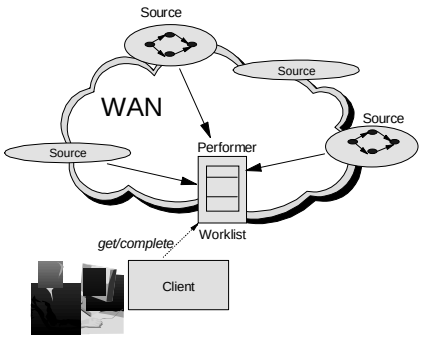


Figure 11: Reusable Worklists

In RainMan, Worklists are treated as addressable network objects, and provide a level of separation between Sources and actual human performers, which makes *asynchronous* exchange of requests and responses between them possible. In other words, requests can be posted to a Worklist on the network even when the human performer is not connected, and the human can access and perform them without connecting to the Source. Most importantly, in contrast to traditional workflow systems, a RainMan Worklist is a first class entity that can be reused by multiple Sources, thus eliminating the need for explicit, dedicated connections to workflow servers on the part of the performer (see Figure 11). This is in sharp contrast to the architecture shown in Figure 4.

Worklists on the network are managed by an independent RainMan Worklist Service. At the present time, the Worklist Service consists of a single Java application on the network, called a Worklist Server, that manages a large pool of Worklists. The Worklist Server is analogous to a POP [Myers96] or an IMAP [Cris93] server that stores e-mail boxes for multiple users. The clear separation of the Worklist Service from the Workflow Server is a novel design point in RainMan. This is useful because the Worklist Service is now streamlined to perform a dedicated function in an autonomous manner, and a Worklist Server can run on dedicated powerful computational resources that guarantee performance, availability, security, and reliability. Furthermore, since Worklists are practically independent of each other, we expect it to be relatively easy in this architecture to address scalability in terms of distributed workflow participants by implementing the Worklist Service as a distributed service; new Worklist Servers can be added to the network to host Worklists as the number of participants increases. We plan to experiment with these issues in the near future.

### 5.2.3 RainMan Performers

The Performer abstraction is useful in modeling humans, software applications, groups and roles, workflow servers, and entire organizations that perform Tasks on behalf of a workflow (see Figure 12). As we have seen, Worklists act as Performers for humans. In addition, we are building a host of other Performers that can be used by RainMan workflows. An interesting Performer class is the `SMTPGatewayPerformer` that allows RainMan workflows to send out e-mail over the Internet. This Performer is an SMTP client written in Java that implements the *PerformerAgent* interface. It can receive *SendEmail* Task requests from Sources, connect to a SMTP server, and submit outgoing e-mail requests. Another implemented Performer class is the `DatabaseQueryPerformer` that can receive SQL queries from a Source and interface with a relational database at the back end via JDBC. A `PalmPilotPerformer` class has also been implemented that allows Worklist access from a US Robotics Palm Pilot.

With the growing use of cell phones, pagers, and PDAs, it is conceivable that a Performer and its human participant may communicate via other metaphors such as *publish/subscribe* (also known as *observable/ observer* or *model/view*), and its variations. The Performer and the human may be co-located on a single machine, or communicate over distances via a wide variety of networks (e.g., wireless, infrared, and so on).

In workflows, groups and roles are used to distribute Task requests to participants according to certain policies. The commonly supported scenario in current workflow systems is the case where an Activity is assigned to all members of a role, say insurance underwriters, and once a role member assumes responsibility for the Activity, it is retracted from the other role members. In RainMan, we take the view that a wide variety of distribution and retraction policies may be meaningful, depending on the application. For example, consider a company that wishes to post a *Request_for_Quotation* to each of its suppliers (these can be modeled as Performers). The company may wish to get results back from each of these suppliers before it can proceed with the next step in its workflow application. Alternately, it may just wish to receive responses from a 'majority' of its suppliers. We are designing concrete classes for Performers that implement such Task distribution and retraction policies based on roles; additional classes can be implemented based on the needs of specific applications.

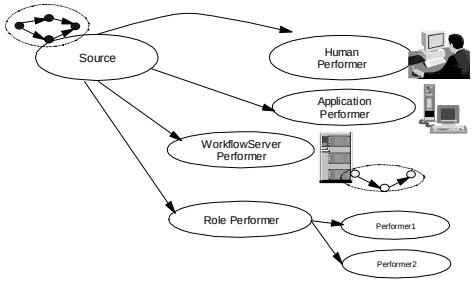


Figure 12: Heterogeneous Performers

# 6. Considerations in Workflow System Design

## 6.1 Performance

To the best of our knowledge, there are no published data on the performance and scalability of commercially available, proprietary workflow systems. However, with the increasing deployment of these workflow systems, concerns have been raised about the lack of adequate performance and scalability in even 'production' workflow systems (an industry label for workflow systems that specifically address the automation of high-volume, highly repetitive processes such as those in banks and insurance companies).

Our view is that the Internet will further aggravate the performance problems of centralized workflow architectures as they try to interoperate. WfMC-compliant, monolithic workflow servers are

designed to handle process management, worklist management, auditing, and directory services simultaneously for hundreds of workflow instances at any given time. This architecture may perform acceptably as long as the number of participants is limited. However, the performance will degrade rapidly as the number of participants grows large.

The field of workflow urgently needs meaningful performance benchmarks that can be used to evaluate existing systems and their architectures. Until such benchmarks are established, it is not meaningful to compare quantitatively RainMan's performance with that of traditional architectures. However, we believe that a compelling qualitative argument in favor of RainMan can still be made. The RainMan design dismantles the traditional monolithic server into a collection of related components. This can help in eliminating the potential performance bottlenecks of workflow servers. For example, the clear separation between process management (Source-side) and worklist management (Performer-side) will allow each of these to be optimized independently. As more human Performers join the RainMan system, additional Worklist Servers can be pressed into service at different parts of the network. In a traditional system, additional worklists would all be added on the central workflow server, thus burdening the server itself. In contrast, the RainMan approach has no impact on the Sources and their performance.

## 6.2 Failure Handling and Compensations

Much of workflow research in recent years has focused on the transactional aspects of workflows [Rus94, Al96, Ley95]. The basic objective is to ensure the recoverability of workflows, since workflows are long-running applications that can execute over days, weeks, or even months. It is fairly well-accepted in database transactions literature that classical flat ACID transactions are not viable in the context of long-running applications. Workflow researchers have thus borrowed concepts from nested transactions, sagas, and spheres of compensation to address the needs of workflows. While many of these ideas remain untested in commercial systems, they appear to be viable in the context of commercial workflow systems of today where the workflow server can exercise significant control over workflow participants.

The decentralization of the workflows, as proposed in RainMan, alters the picture significantly. In particular, if Source workflows are to execute on the Internet, the autonomy of Performers and their non-proximity to each other and to the Source itself must be taken as a given. A Source in this context has no global authority or jurisdiction over remote, heterogeneous, autonomous Performers, and has very limited visibility of their internal resources and mechanisms. In effect, all notions of compensation must be addressed in terms of commitments between peers; in effect, the interaction between a Source and a Performer must be sufficiently rich to handle failures and provide compensations for past services as necessary. To use a classical example, a Source may be a Travel Booking workflow that interacts with a Performer such as a Hotel Server. The Hotel Server may provide two basic operations - *reserve* and *cancel* - where *cancel* is a compensation for a *reserve*. Because of a subsequent change in travel plans, the Source may return to the Performer with a *cancel* request on its past *reserve* request. The Hotel Server, within the limits of its service contract, would have to fulfill this request. The idea of long-running *conversations* between autonomous network entities that engage in business transactions has been explored in the Coyote project [Dan97]. The Coyote view that meaningful business transactions can occur despite limited authority of each participant is a useful one for the Internet; we are exploring how RainMan Sources and Performers can benefit from the idea of conversations.

## 6.3 Decentralized Execution

Traditional workflow systems are based on a model of centralized workflow execution; the workflow system is responsible for managing workflow coordination as well as activity execution by invoking resources or participants entirely within its scope of authority - applications, other workflow servers, or human worklists.

A diametrically opposite model of workflow execution that can decentralize both workflow coordination and activity execution has been proposed in the context of the Arjuna project [Ran97]. This execution model decentralizes the coordination of a process by installing 'task controller' objects in different domains that coordinate with each other to deliver workflow routing functionality. Each task controller is a workflow 'router' that understands a piece of the overall workflow graph. This execution model eliminates a central point of failure in a workflow; moreover, workflows can proceed even in the face of partial network failures. The main consequence of the Arjuna approach is that decentralized workflow control requires participant domains (i.e. service provider domains) to participate in workflow routing on behalf of the workflow using a pre-agreed coordination language (a workflow routing protocol); this imposes computational burdens on participant domains that would be unacceptable if the

domains were autonomous. Decentralized control can also be expensive to manage; it is harder to maintain global state and make dynamic changes to the workflow when the workflow script itself is decentralized.

The RainMan execution model strikes a middle ground. It separates the responsibility of workflow coordination from activity execution by creating two classes of entities, Sources and Performers. In effect, while the coordination of each process remains localized within a Source object, the actual execution of activities is decentralized across a network of Performers over which Sources have very limited control. The leverage in this model arises from the ability of heterogeneous Sources to share heterogeneous, autonomous Performers. This approach respects the autonomy of each Performer, assumes that the environments in which Sources and Performers execute will necessarily be heterogeneous, and makes it easier to keep track of global state as well as support dynamic workflow modifications.

### 6.4 Security Considerations

For workflows to run across wide area networks and especially across organizations, multiple security concerns must be addressed. First, an authentication mechanism must exist to validate the identity of both Source and Performer domains. This would allow basic functions such as Worklist access and Performer invocation to be done in a secure fashion only by authorized users or components. Second, access control rights need to be described and enforced in a scalable fashion to control access to methods on Performers and Sources. Third, the integrity and privacy of Task requests and responses exchanged between Sources and Performers should be maintained. Finally, support for nonrepudiability and enforcement of terms and conditions is needed. We are currently exploring these issues by drawing from the state-of-practice in distributed systems security. Many of these problems can be easily alleviated in the case of workflows between trusted parties by setting up private channels (Intranets or Extranets) between the participant individuals and organizations.

## 7. Conclusions

Our research is directed at designing an Internet workflow infrastructure that is scalable, flexible, and interoperable. This is a relevant and important problem since individuals and organizations are rapidly getting interconnected. This widespread interconnectivity can be exploited to enable new kinds of process-based applications.

The RainMaker framework defines the essential abstractions of a workflow system and facilitates interpretable workflow components. Using the RainMaker framework, we have implemented RainMan, a distributed, object-oriented workflow system written in Java. Workflow management, activity distribution, directory services, and worklist management are all treated as independent services that work together to deliver workflow functionality to Internet users. This is a radical departure from traditional workflow systems based on monolithic, server-centric architectures. The RainMan system uses open standards and Web-browser based user interface components. The system is being used to experiment with a range of interesting features such as decentralized workflow execution, dynamic workflow modification, and disconnected participation.

While RainMan has been designed as an infrastructure for workflow execution, it offers insights into the broader problem of designing long-running applications on a network. In effect, RainMan highlights the importance of separating the responsibilities of service requesters (in this case, workflows) from service providers (in this case, humans, applications, organizations, etc.) via clean interfaces (i.e. *SourceAgent* and *PerformerAgent*), and assuming that entities that implement these interfaces are heterogeneous and autonomous. It offers a peer-to-peer Task delegation model with a nice recursive behavior; a Performer that receives Task requests from a Source can itself act as a Source of Tasks for other Performers on the network.

## 8. Acknowledgments

We thank David Hutches and Sastry Duri for various discussions on workflows and compensations. We also thank Carl Staelin and the reviewers of this paper for their valuable comments.

## References


[ActWork] Action Technologies, Action Workflow, *http://www.actiontech.com*

[Al96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor, and C. Mohan, Advanced Transactional Models in Workflow Contexts, In Proceedings of ICDE, 1996.

[CCITT] CCITT/ISO, X.500, The Directory - Overview of Concepts, Models and Services, CCITT/ISO IS 9594.

[Cris93] M. Crispin, IETF RFC 2060, Internet Message Access Protocol version 4 rev 1, December 1993

[Dan97] A. Dan, and F. Parr, The Coyote Approach to Network Centric Service Applications, 7th International Workshop on High Performance Transaction Systems, Asilomar, California, September 14-17, 1997.

[FlMa] IBM Corporation, FlowMark Workflow, *http://www.software.ibm.com/ad/flowmark*

[Hama96] M. Hamalainen, A. B. Whinston, and S. Vishik, Electronic Markets for Learning: Education Brokerages on the Internet, CACM, Vol. 39, Number 6, June 1996.

[InConc] InConcert Inc., InConcert Workflow, *http://www.inconcert.com*

[Ley95] F. Leymann, Supporting Business Transactions via Partial Backward Recovery in Workflow Management, in Proceedings of BTW'95, Dresden, Germany, 1995, Springer Verlag.

[Mock87] P. Mockapetris, IETF RFC 1034/1035, Domain Names - Concepts and Facilities, Implementation and Specification, November 1987.

[Myers96] J. Myers, and M. Rose, IETF RFC 1939, Post Office Protocol - version 3, May 1996.

[OMG97] OMG Trading Object Service, CORBA Services: Common Object Services Specification, Chapter 16, July 1997.

[Paula97] S. Paul, E. Park, and J. Chaar, Essential Requirements for a Workflow Standard, OOPSLA Workshop on Business Objects Design & Implementation, October 6th, 1997, *http://www.tiac.net/users/jsuth/oopsla97*

[Paulb97] S. Paul, E. Park, D. Hutches, and J. Chaar, RainMaker: Workflow Execution Using Distributed, Interoperable Components, IBM Research Report nbr. 21008, October 1997.

[Ran97] F. Ranno, S.K. Shrivastava and S.M. Wheater, A System for Specifying and Coordinating the Execution of Reliable Distributed Applications, International Working Conference on Distributed Applications and Interoperable Systems (DAIS'97), September 1997.

[Rus94] M. Rusinkiewicz and A. Sheth, Specification and Execution of Transactional Workflows, In W. Kim, Editor, Modern Database Systems: The Object Model, Interopreability and Beyond, ACM Press, 1994.

[Schu96] W. Schulze, M. Bohm, and K. Meyer-Wegener, Services of Workflow Objects and Workflow Meta-objects in OMG compliant Environments, OOPSLA Workshop on Business Objects Design and Implementation, 1996.

[Silver95] B. Silver, The BIS Guide to Workflow Software, BIS Strategy Decisions, One Longwater Circle, Norwell, MA 02061, 1995.

[VisFlo] FileNet Corporation, FileNet Visual Workflo, *http://www.filenet.com/products/vwtext.html*

[WfMC] Workflow Management Coalition, *http://www.aiai.ed.ac.uk/WfMC*

[Yeong95] W. Yeong, T. Howes, and S. Kille, IETF RFC 1777, Light Weight Directory Access, March 1995.