ELSEVIER

Decision Support Systems 32 (2002) 265–278

Decision Support Systems

www.elsevier.com/locate/dsw

Workflow support for electronic commerce applications ☆

Akhil Kumar [a,*,1], J. Leon Zhao [b]

[a] College of Business, Campus Box 419, University of Colorado, Boulder, CO 80309-0419, USA
[b] Eller College of Business and Public Administration, University of Arizona, Tucson, AZ 85721, USA

Received 22 November 2000; accepted 6 January 2001

Abstract

Internet-based electronic commerce is becoming the next frontier of new business opportunities. However, commerce on the Internet is seriously hindered by the lack of a common language for collaborative commercial activities. Although Extensible Markup Language (XML) allows trading partners to exchange semantic information electronically, it does not provide support for document routing. In this paper, we describe various inter-organizational electronic commerce applications and discuss their needs for workflow support. Then, we propose a blueprint for XRL, an Extensible Routing Language that enables routing of commercial documents over the Internet and helps in creating truly intelligent documents. This routing language is simple, yet powerful enough to support flexible routing of documents in the Internet environment. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Inter-organizational electronic commerce; EDI; B-to-B and B-to-C commerce; Extensible routing; Document management architecture; XRL; XML

1. Introduction

The Internet and the World Wide Web (WWW) have changed the landscape of networked computing and have become the de facto environment for electronic commerce. However, the current electronic commerce technologies rely on much in-house programming activities and are inefficient and lack interoperability. The trend is to develop more standardized architectures and techniques for open electronic commerce services [4,5,11,17]. One important thrust for increased productivity and interoperability is to develop more homogeneous languages for various electronic commerce activities [9]. A second thrust is towards developing autonomous, cooperating agents that can communicate intelligently with one another [16]. In this paper, we focus on the development of a language that provides support for routing of workflow for Internet-based electronic commerce services. In the spirit of HTML which provides support for rendering documents in a platform independent manner, and XML (Extensible Markup Lan-

☆ An earlier version of this paper appeared in the International Conference on Telecommunications and Electronic Commerce, Nashville, TN, October 1998.

* Corresponding author. Bell Labs, 600 Mountain Avenue, Rm. 2A-406, Murray Hill, NJ 07974, USA.
E-mail addresses: akhil@acm.org (A. Kumar), Lzhao@bpa.arizona.edu (J. Leon Zhao).

[1] Currently on leave as a visiting researcher at Bell Labs, Murray Hill, NJ. This author's work was supported by the Research Committee of the College of Business, University of Colorado, and by the David Lattanze Center for Executive Studies in Information Systems, Maryland.

0167-9236/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-9236(01)00114-2

guage) which allows exchange of semantic information, the proposed language called Extensible Routing Language (XRL) provides support for routing documents and managing workflows across trading partners.

The Web has evolved through various stages. It started as a way of accessing distributed information on the Internet using a GUI interface based on the HTTP protocol and the HTML language. It was a successor to Gopher, WAIS and Archie, which were also information access and knowledge discovery tools but were based on text-based interfaces [13]. The next important advancement was to make the HTML protocol more interactive using the FORMS feature so that commercial tasks such as buying and selling, filling in surveys, searching databases, etc., could also be performed. In 1997, about US$40 billion worth of transactions were performed over the Web. This figure has grown exponentially since then, and is now around several hundred billion dollars. Several architectures for Web commerce have been proposed, e.g. Commerce Net, etc. [10] and protocols like SSL and SET [6] have been developed to address security issues that arise on the Web.

Table 1 summarizes the main stages through which the web has evolved. Stages 2 and 3 reflect the current stage of development of the web. XML [19,20] makes it possible to add semantic information to an HTML document so that trading partners can understand the meaning of various fields in the document. Fig. 1 shows a common model of how the web operates to provide interactive services such as shopping, database access, etc. The various steps are as follows:

(1,2) A client connects to the server and downloads a form.

Table 1
Stages of Web evolution

Stage 0: Gopher, WAIS and Archie [8] were primitive, text-based tools for knowledge discovery.
Stage 1: Web as a hypertext navigation tool for knowledge discovery using GUI interface.
Stage 2: Web users could interact with a server program and databases using CGI [6] gateways (see Fig. 1).
Stage 3: Asynchronous mode interaction between a series of trading parties using a semantic language XML [12].
Stage 4: Add workflow features with routing semantics and enable interoperability.

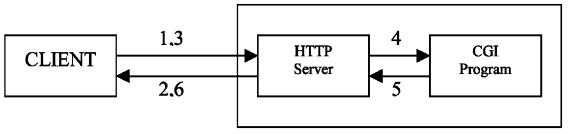


Fig. 1. Existing model of the Web using CGI-gateways.

(3) The client fills in information on the form and submits it.
(4,5) A CGI program on the server processes the form.
(6) The server sends a reply to the client.

The features of this model are that all interaction is in *synchronous* (request, reply) mode.

While it works very well for a variety of applications, there are other instances where it is not so effective. In particular, some of the problems are as follows. First, it is not always possible to establish a connection between the client and server either because the underlying network is unavailable or the server is overloaded. Secondly, this is not a very efficient way of communication when a large amount of information has to be exchanged. Thirdly, this form of communication is not very conducive to *flow-type applications*, which involve a flow of information/documents through multiple workers in several organizations. Thus, there is a need for a framework that exploits the advantages of the web, and yet provides better support for *asynchronous*, *flow-type* commercial applications. This kind of situation arises frequently in supply chain management [2,3,15]. In such situations, asynchronous transmission of messages can be more reliable and efficient. For instance, it is considerably easier and more efficient for a customer to place large numbers of orders (for stocks, etc.) or make bids in an electronic market (for several products at the same time) asynchronously.

This paper is organized as follows. Section 2 discusses the need for workflow support in Electronic Commerce. Section 3 gives an overview of the intelligent document architecture based on routing slips. Next, Section 4 describes the XRL language. Then, Section 5 presents a preliminary design of the document routing assistant based on the XRL language. Section 6 discusses various theoretical issues

that arise in the context of this architecture, the language, and the system. Finally, Section 7 concludes the paper.

2. Electronic commerce scenarios

2.1. Electronic commerce and workflow automation

Electronic commerce requires workflow support. Actually, electronic commerce has been around for many years in the form of Electronic Data Interchange (EDI) . However, EDI is limited to business-to-business (B-to-B) commerce and is based on structured document formats (called transaction sets), which are transmitted over value-added networks. Electronic commerce over the Internet [18], or Internet commerce, has opened up new possibilities of business-to-consumer (B-to-C) commerce by obviating the need for value added networks. In the first step, the Internet serves as a router for communications between trading partners, but more importantly, in a broader sense it can lead to the creation of new types of value chains between partners. An electronic commerce application must also provide support for *notification* and *negotiation* [22]. Parties must be able to negotiate before reaching a commitment, on price, delivery date, etc. Notification is another important feature. After a customer places an online trade, the broker should notify him or her as soon as the trade is executed rather than the customer checking the web site continuously to find out if the order has been executed. Similarly, a package delivery service such as Federal Express should notify the customer (both sender and receiver) as soon as a package is delivered at a site. All these electronic commerce activities point to the importance of workflow automation as further exemplified in the next few subsections.

2.2. Workflow in manufacturing

A multi-level supply chain [3] is shown in Fig. 2. At the enterprise level, the logistics agent interacts with the customer about an order. To handle this order, logistics has to decompose it into activities such as manufacturing, assembly, transportation, etc. Then it negotiates the execution of these activities with the available plants, suppliers, and transportation companies. At the plant level, a selected plant will similarly plan its activities including purchasing materials, using existing inventory, scheduling machines on the shop floor, etc. Additional negotiations and other type of exchanges between the enterprise and plants are often necessitated due to unexpected events and breakdowns.

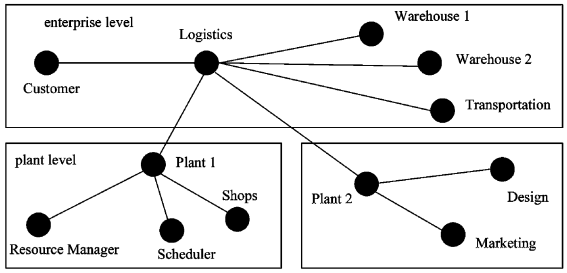


Fig. 2. A multi-level supply chain.

2.3. Workflow in mail-order processing

Consider that a customer places an order for four books with Amazon.com. Amazon in turn places four different orders (one for each book) with the respective publishers, who send their books separately to a shipper. The shipper coordinates the receipt of the four different shipments from the publishers and makes one package to be shipped to the customer. In this simple example, there are seven parties who have to coordinate among themselves in order to successfully complete the order. Even during the order processing phase, there are several complications that can arise:

- The customer may cancel or change the order.
- A publisher may not have a book in stock.
- A publisher may delay shipment.

These are called *exceptions* and numerous other such abnormal situations can arise [21]. In all these scenarios, there is a need for additional coordination and, possibly negotiation.

2.4. Workflow in health care

Fig. 3 shows an example of workflow in Healthcare. A patient with an ankle injury visits a primary

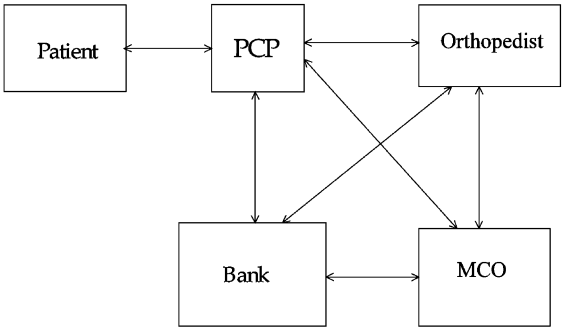


Fig. 3. Workflow in healthcare.

care physician (PCP). The PCP decides to refer the patient to an orthopedist. In order to find an approved orthopedist, the PCP must consult with the managed care organization (MCO) and arrange to deliver the patient's document to the selected orthopedist. Eventually, the payment has to be coordinated between the MCO, PCP, orthopedist and one or more banks.

3. Intelligent document management architecture

There are two architectures for workflow management systems. In the *centralized* architecture, which is more common, there is a central server that maintains the workflow repository and coordinates the workflow processes in the system, and the client provides the interface to the workflow management system and other software tools. On the other hand, in a fully *distributed* architecture, every node is a fully functional subsystem, and there is no central node that contains all information on the workflow processes in the system [1]. Our proposal for intelligent documents is most suitable with this kind of architecture (see Fig. 4). A role in this figure represents a worker who performs a task. For example, a *manager* is a role denoting a set of tasks that a manager can perform. Examples of other roles are customer, clerk, inspector, vice-president, etc.

Our general approach towards developing *intelligent documents* can be described with the notion of a *routing slip*, to which one or more documents are attached as shown in Fig. 4. Workflow servers exchange information in this manner. Moreover, Fig. 4 also shows that when a workflow server receives a routing slip along with document(s), it will in turn send a reply or confirmation back to the sender. This confirmation serves as a receipt (or handle) for querying purposes and by following a chain of such receipts it is possible to determine the exact status of the workflow in a distributed environment. Next, we develop the idea of a routing slip in more detail since it is an important component of our architecture.

A *routing slip* is a simple sequence of addresses (or users) to which a document must be sent. One or more documents can be attached to, modified, and detached from the routing slip by various workers (or nodes) on the slip. The routing slip is created by an owner of the slip who defines the routing pattern and specifies permissions in terms of who can make what changes (if any) to the routing slip. It has a unique ID that can be used to trace the routing slip. A routing slip can be described or specified using XRL and stored in a file called the document routing definition (or DRD) file. It may also be embedded in a document. However, if it is stored in a separate file then it is possible to attach several documents to it. The routing slip is used as follows.

(1) The workflow originator initiates the document and attaches a routing slip to it. He or she starts the work on the document and specifies the contents and sequence of work by workers involved in preparing and approving the document by selecting and/or modifying the routing slip.

(2) The workflow originator then routes the document to the next worker through email.

(3) The next worker then completes the job assigned to her, and updates the routing slip appropriately to indicate the progress made by her. The

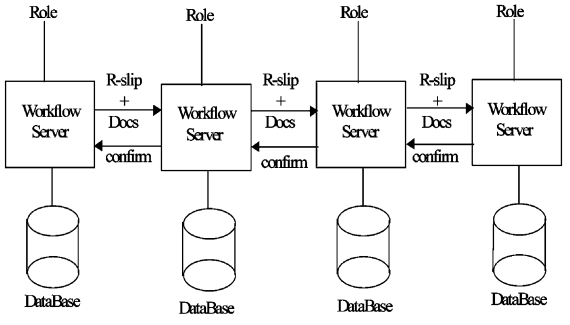
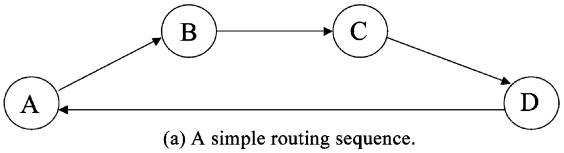
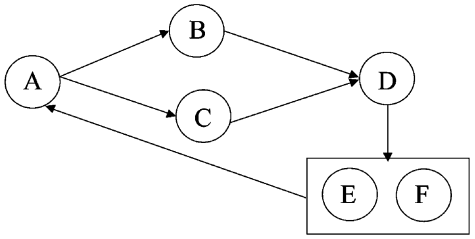


Fig. 4. A distributed architecture for an interorganizational workflow system.



(a) A simple routing sequence.



(b) A complex routing sequence.

Fig. 5. Simple and complex document routing.

specific task of the worker may be to add to, modify, comment on, or approve the document.

(4) In general, the routing may be simple or complex as illustrated in Fig. 5. Fig. 5(a) illustrates a simple route that involves a sequential process with tasks A, B, C, D and, finally A, again, in that order. Fig. 5(b) shows a complex route that includes a *parallel process* and a *flexible sequential process*. Following A, B and C may occur in parallel. The flexible sequence lies in the path from D to A, where E and F may occur in any order, but not in parallel.

(5) The routing slip may also have additional information such as time stamps (i.e., when it was sent), identity of workers who performed tasks on it, and the number of times it was worked on. By looking at the routing slip it should be possible to determine the entire history of the documents attached to it.

The routing slip will be used along with a routing facility, called the *document routing assistant* (DRA), which is a lightweight workflow server as described in Section 5.

4. Overview of XRL

XRL is a way to embed routing information in a document so that it can be routed in a variety of different ways. We first define *straight sequence*, *parallel*, and *flexible sequence* routing. These basic constructs can then be combined together to develop more complex routing schemes. The theoretical basis for these routing blocks can be found in the authors' related work on workflow models [14].

4.1. Straight sequence

The simple routing pattern is sequential. Here is an example of how a sequential route can be specified using XRL.

```
⟨?XRL version = 1.0
owner = "host1:user1" ID = id1 ⟩
⟨ROUTE ⟩
host1:user1 ⟨attribute1 = x1, attribute2 = x2 ⟩
host2:user2. . .
⟨/ROUTE ⟩
```

This describes the sequence that a document must follow. Each entry in the routing slip consists of a host or server address, a user address and some attributes. It should be noted that the user address could either be a generic role (e.g. "order_clerk") or the address of a specific user (e.g. "john"). The server address is of course mandatory. However, if the user address is omitted, then the document is routed to a default user. Similarly, if the attributes are omitted, then default values for them are assumed. The various attributes will be described shortly. For convenience, the construct of ⟨ROUTE ⟩. . .⟨/ROUTE ⟩ is referred to as a *routing block*.

Example 1. This example illustrates how, after a customer places an order on the vendor's web site, it is routed through the orders department of Lucent, then to the manufacturing department, then to billing, and finally a confirmation is sent to the customer.

```
⟨?XRL version = 1.0
owner = "lucent.com:orders" ID = id1 ⟩
⟨ROUTE ⟩
lucent.com:orders⟨Attach = order.html ⟩
lucent.com:manufacturing
lucent.com:billing
lucent.com:shipping
⟨Detach = order.html, Attach = confirm.html ⟩
compuserve.com:Akhil
⟨/ROUTE ⟩
```

The order.html document (or order "form") is *attached* at the first node (i.e. the orders department) and then it is routed to manufacturing, billing and shipping in sequence. Each node will add appropriate information along the way. Finally, the shipping department will *detach* the order "form" from the routing slip and *attach* a confirmation form and send it to the customer. *Detach* and *attach* are two attributes associated with each node on the routing slip.

4.2. Parallel split

ROUTE SPLIT allows a document to visit any m of the n nodes in parallel ($m < n$). In general, these m nodes can be determined arbitrarily or according to some considerations like availability of the users, etc. The routing schema, per say, does not specify a criterion.

⟨?XRL version = 1.0
Owner = "host0:user0" ID = id1 ⟩
⟨ROUTE SPLIT *m* of *n* ⟩
host1:user1
host2:user2. . .
⟨/ROUTE ⟩

Example 2. To express the rule that a payment invoice must go to *any two* of the *three* vice-presidents for approval and the approvals can occur in parallel, we write:

⟨?XRL version = 1.0
Owner = "abc.com:sales" ID = id1 ⟩
⟨ROUTE SPLIT 2 of 3 ⟩
abc.com:vp-finance
abc.com:vp-sales
abc.com:vp-accounts
⟨/ROUTE ⟩

4.3. Parallel join

Note that ROUTE SPLIT and ROUTE JOIN blocks must appear in pairs. A Route Join block is shown below.

⟨ROUTE JOIN *m* ⟩
host:user
⟨/ROUTE ⟩

This syntax means that m splits are *joined* at the node or server denoted by 'host'. The workflow system will have to coordinate the join activity and ensure that the same m branches, which were split up earlier are subsequently joined. The next example gives an illustration.

Example 3. For instance, the next JOIN block can be combined with the SPLIT block in Example 2 above.

⟨ROUTE JOIN 2 ⟩
abc.com:accounts
⟨/ROUTE ⟩

4.4. Flexible sequence

The notion of flexible sequence is defined as follows:

⟨?XRL version = 1.0
Owner = "host0:user0" ID = id1 ⟩
⟨ROUTE SEQ *m* of *n* ⟩
host1:user1
host2:user2. . .
⟨/ROUTE ⟩

The ROUTE SEQ structure requires that a document must visit *any* m of the listed n nodes ($m < n$) in *any* sequence (but *not* in parallel). That is, the document must be routed from one to another recipient until m of them have seen the document, but it cannot be routed to more than one recipient at the same time. Note that unlike a SPLIT block, it is not required that a SEQ block should be followed by a corresponding JOIN.

Example 4. In this example, an invoice requires approval from *any two* of the *three* vice-presidents,

and these approvals may take place in *any* sequence (but not in parallel).

```
⟨?XRL version = 1.0
Owner = "abc.com:sales" ID = id1 ⟩
⟨ROUTE SEQ 2 of 3 ⟩
abc.com:vp-finance
abc.com:vp-sales
abc.com:vp-accounts
⟨/ROUTE ⟩
```

4.5. Nested routing slips

Example 5. In the following example, a ROUTE SPLIT block is nested inside a sequential ROUTE block. In this routing pattern, a document is first sent to the accounts department, then routed in parallel to the three vice-presidents, and finally routed back to accounts, where it is merged.

```
⟨ROUTE ⟩
abc.com: accounts
    ⟨ROUTE SPLIT 2 of 3 ⟩
    abc.com:vp-finance
    abc.com:vp-sales
    abc.com:vp-accounts
    ⟨/ROUTE ⟩
    ⟨ROUTE JOIN 2 ⟩
    abc.com:accounts
    ⟨/ROUTE ⟩
⟨/ROUTE ⟩
```

The *JOIN* block in this example involves a merge of two documents. If two documents write to the same field, then upon merging one value will be overwritten by the other. Assuming that the same field of the document will not be written independently along the two paths, this will not cause any conflicts. Therefore, the parallel paths have to be designed appropriately. On the other hand, if both the paths were allowed to write to a field then the semantics can become unpredictable, and normally the application would disallow it.

4.6. Routing based on conditions

Consider the following sequence of interactions: A customer orders a product. The order department checks that the product is available and confirms the order to the customer. The order is then passed on to the shipping and accounts departments. If the product is out of stock, then the order department notifies the customer.

```
⟨?XRL version = 1.0
owner = "lucent.com:orders" ⟩
  ⟨ROUTE ⟩
  lucent.com:orders
    ⟨ROUTE CONDITION
    status = "out-of-stock" ⟩
    ⟨TRUE ⟩
    compuserve.com:Akhil
    ⟨FALSE ⟩
    lucent.com:mfg
    lucent.com:billing
    lucent.com:shipping
    compuserve.com:Akhil
    ⟨/ROUTE ⟩
  ⟨/ROUTE ⟩
```

In this example, *status* is a field in the associated document already defined using XML. If several documents are attached to the routing slip, then using dot notation, i.e. *doc-title.field-name* can denote a specific field of a document.

4.7. Exception handling and routing

Sometimes a need for exception or ad hoc routing can arise. In such a situation, the document is often returned to an earlier stage. This would be handled by overriding the routing slip and sending the document back to an earlier stage. This means that the operations performed in the intermediate stages must then be undone, and redone subsequently during rework. Therefore, it is necessary that any changes that are made to the document be time stamped and historical values be retained.

Historical data is of two types, *document history* and *routing history*. A *document history* logs changes to various fields in the document. A document log is

similar to a transaction log in a database system and can be stored in a format such as ⟨worker ID, document ID, field, old value, new value, time stamp⟩. *Routing history* contains information about access to the document by various workers and can be recorded as ⟨document ID, worker ID, time-stamp⟩. Note that there is an overlap of information between a document history and a routing history. However, both are needed for two reasons: (1) A document history only logs modifications to the document, and does not contain information on read access. (2) A routing history is more efficient to use than the document history because it is more compact.

4.8. Attributes

Table 2 gives a list of various attributes used in the paper along with their descriptions and default values. The *attach* and *detach* attributes are self-explanatory. The *rework* attribute specifies whether a node is allowed to send the document to a previous node for corrections. The delay attribute specifies the maximum amount of time (say, in minutes) a node can keep a document. The *change* attribute specifies whether a node can change the routing slip. The *confirm* attribute states whether a step on the routing slip is a confirmation step (see further discussion of this in Section 6.2). The *abort* attribute determines if a node may abort the routing slip (see subsequent discussion in Section 6.5).

Table 2
Various attribute names and their default values

Name	Description	Default value
Attach	Name of another document to be attached	No new attachments
Detach	Name of a document to be detached	None
Rework	Can the document be routed back to another node?	"No"
Delay	Maximum amount of time a node can work on the attached document	No limit
Change	Can the node change the routing slip?	"No"
Confirm	Is this a confirmation step?	"No"
Abort	Can the node *abort* the routing slip?	"No"

5. Analysis and design of the document routing assistant

In this section, we present a preliminary design of a lightweight workflow server called a document routing assistant (DRA), which is located at each client node. In the architecture of Fig. 4, some of the server nodes may be viewed as DRA nodes, while others are full-fledged workflow servers called primary servers. This paper focuses mainly on the DRA design because it provides all the routing capabilities, which is the major theme of this paper. The DRA provides basic routing functionality and also interacts with one or more primary workflow servers, which store the databases and have additional capabilities. The following discussion covers an analysis of document routing activities, a system architecture of the DRA, its main system modules, and the DRA's handling of typical routing scenarios. We then discuss how the DRA can be used in a client-server setting.

5.1. Basic document routing activities

The DRA should help the worker with two basic document routing activities, document receiving and document forwarding. The tasks associated with these two activities are discussed below.

- *Document receiving*: When a new message arrives at the worker's site via the Internet, the DRA should first determine if the message contains any workflow tasks and whether or not these tasks can be started. A piece of workflow task can be started when all its preconditions are met, e.g. a task may require documents from multiple senders or a specific timeframe for the task has been set. If there are associated conditions, the DRA must start monitoring them by examining all incoming messages relative to these conditions. In this regard, the DRA is more than just assisting with routing; it is also assisting with management of workflow processes. Once the workflow task is ready to start, the DRA prepares the working environment, e.g., by decrypting the documents if they are encrypted. Note that by

examining the workflow conditions and monitoring the arrival of messages and documents related to the task, the DRA adds a great deal of value to the worker by relieving the worker from the tedious tasks of comparing workflow procedures with messaging events.

- *Document forwarding*: Once the worker completes a task, the DRA routes any completed documents and sends information on the work status of the documents further. For example, a worker may be required to acknowledge the receipt of a document or to confirm that he (she) has read the document(s). Sometimes, multiple messages may have to be sent after receiving a document such as acknowledging the receipt to the sender and at the same time forwarding the document to the next worker(s) in the workflow after completion of a task. The DRA can offer help by interpreting the routing slip and sending the required messages at the worker's request. The worker no longer has to manage the real addresses of the message recipients or to send each message individually.

5.2. A system design of document routing assistant

A system design of the DRA is presented next. The DRA provides a mechanism to operationalize the XRL language. Fig. 6 illustrates a design of the DRA that consists of eight modules described next.

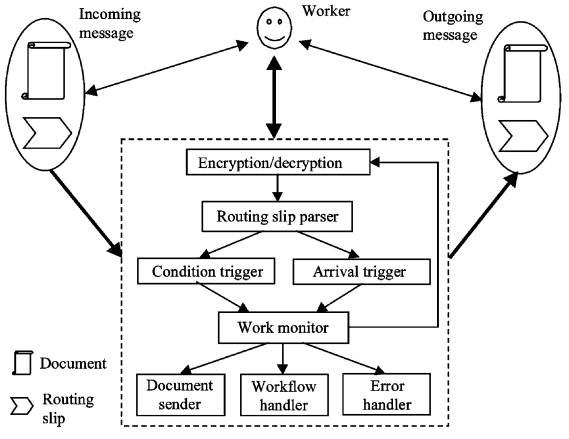


Fig. 6. A high-level system design of the document routing assistant (DRA).

Note that the arrows in the figure indicate a possible order of execution of modules for a workflow instance. Other ways of organizing and linking the boxes are also possible. The DRA will interface with other DRAs using standard message access paradigms, and protocols like Internet Message Access Protocol (IMAP) and Post Office Protocol (POP).

To accomplish the basic routing activities defined above, namely document receiving and forwarding, the DRA contains the following system modules.

- *Encryption/Decryption*: For security reasons, the system must be able to decrypt and encrypt documents and routing slips. Depending on the level of security requirements, either symmetric or asymmetric keys may be employed. While asymmetric keys usually provide a higher level of security, they are more costly to use and manage.
- *Routing Slip Parser*: The parser extracts the routing blocks in the routing slip relevant to the current worker and prepares the data for use in other system modules.
- *Routing Condition Trigger*: If a task in the routing slip contains conditions, a routing condition trigger must be set up to monitor the condition. The condition may be associated with either document arrival or forwarding.
- *Document Arrival Trigger*: In case a task requires documents from multiple messages, a document arrival trigger will turn on and poll the messaging system for relevant messages related to the task. This monitor must understand the local messaging system and maintain the routing condition trigger.
- *Work Monitor*: The routing slip may also include information on the status of the worker and the associated constraints that the work is subject to when working on the document. For instance, a certain worker may not be allowed to modify particular fields of the document, and only a particular worker such as the owner of the workflow may modify the routing slip. The work monitor is a software module that performs such integrity checks on the operations performed by the worker on the document and the routing slip.
- *Document Sender*: At the completion of a task, and after all conditions are met, the document sender simply forwards the documents and routing slip to the next worker(s) in the workflow. If the current node in

the workflow is a split point, multiple messages are sent.

- *Workflow Handler*: Sometimes, a routing process may require updating the workflow status on a main workflow server or the node of the workflow owner. The workflow handler takes charge of such higher level tasks. It may also collect statistics related to the workflow on parameters such as sojourn time at each node.
- *Routing Slip Error Handler*: In case there is an error in the routing slip, the error handler will need to mediate. For instance, there might be a warning message by the local messaging system that the message sent out was not properly executed or the forwarding address did not exist. These types of messages must be properly processed in order to ensure that the workflow will be carried out correctly.

5.3. A functional analysis of the document routing assistant

To show that our design of the DRA is functionally complete, we next analyze how the DRA can support the various routing scenarios of Section 4.

- *Straight Sequence*: If the current routing involves a straight sequence, then the role of the DRA is very simple. After decrypting the document and routing slip, the DRA prompts the worker of the arrival of the task and appends it to a queue. Since there is no condition or join construct involved, the condition and arrival triggers are not set. Once the task is completed, the document sender encrypts the document and routing slip, and forwards them in a message to the next intended receiver.
- *Parallel Split*: In case of a parallel split, the DRA sends the completed document to multiple workers in the workflow. As in the case of straight sequence, no triggers are set.
- *Parallel Join*: When the current node is the location of a parallel join, the DRA sets up the arrival trigger to monitor the arrival of the relevant documents. When *all* documents arrive at the worker's site, the DRA inserts the task in the queue of the worker who is next in the process.
- *Flexible Sequence*: Recall that in a flexible sequence there is a choice of the next destination. Therefore, the DRA prompts the worker with all possible (next) routing destinations and asks the worker to choose only one and then sends the document there. This assumes that the worker has additional information to make such a choice. Of course, another possibility is that the DRA may make a random choice of the next destination among the various possibilities. In addition, the DRA should also update the routing slip to indicate the workers that have already processed the workflow instance.
- *Conditional Routing*: When routing is conditional on information to be entered into the document, the document sender module must retrieve the data from the relevant fields in the document and evaluate the condition. Naturally, the result of the condition evaluation will determine the next destination(s). This can be a bit more involved because, to execute the routing properly, the DRA might need to understand the fields of the document(s) that are referenced in the condition clause.
- *Nested Routing Slips*: A nested routing slip is necessary when the workflow involves conditions, joins, or splits. In these situations, the DRA must be able to parse a nested slip and identify the constructs involved.

We envisage a client-server environment for operating the DRA. The DRA will be developed as a generic desktop software component and installed as an add-on to the client computer. The DRA should be capable of interacting with the default messaging system at the client and also be able to interact with the primary workflow server that manages the workflow processes and the databases. The DRA can use the Internet browsers as the user interface for the worker.

The DRA will be equipped with Open Data Base Connectivity (ODBC) and Java Data Base Connectivity (JDBC) capabilities to access the primary workflow server. Moreover, the DRA might also need to communicate with a primary workflow server for resolving any problems with the routing slip, or for communicating with networks that it does not understand.

6. Other issues related to routing slips

There are several issues of consistency and correctness related to routing slips. These issues are

addressed in this section. More specifically we are concerned with the following points:

What is a *complete* routing slip?
What is a *correct* routing slip?
Who can *modify* the routing slip?
What happens if one of the addresses on the routing slips is *not valid*?
Should *cycles* be allowed in the routing slip?

6.1. Completeness

By completeness, we mean that most common kinds of routing scenarios, which arise in production and ad hoc workflow environments, can be represented correctly and succinctly. A formal proof of correctness is beyond the scope of the present paper. However, completeness can be verified by showing that most routing scenarios can be described using straight sequence, flexible sequence, parallel routing through splits and joins, and conditional routing based on conditions.

Of course, there are certain more complex scenarios that cannot be represented with the language in its present form. For instance, the conditional routing is based on a very simple checking of the value of a field. Thus, with reference to Example 2 from the previous section, it would be hard to specify that "if two out of three VP's approve the invoice then route to accounts department, else route to purchasing department." As always, there is a trade-off between expressive power on the one hand and complexity of the language on the other. For now, we lean towards simplicity in our design. Such additional capability requires a tighter link with the underlying data model.

6.2. Correctness

There are two aspects of correctness, syntactic and semantic correctness. Semantic correctness is hard to verify; therefore, here we concern ourselves mainly with syntactic correctness. This can be stated in terms of rules. Basically, a correct routing slip should have the following features.

(1) All server addresses and user names must be valid along the route. The server addresses can be verified by using a command like "ping", which sends a packet to another machine to see if it is alive, or by performing a name server lookup (*nslookup*) to find its Internet (IP) address. The user address can be verified by using an address resolution mechanism of the underlying messaging system. Once the server and user names are verified as being valid they may be stored for future reference.

(2) There should not be any uncontrolled cycles in the routing pattern. The only cycles that should be allowed are the ones that require a confirmation to be sent back to a node on the routing slip. This is indicated by an attribute *confirm* (see Table 2). Graph-based techniques can be used for cycle detection.

(3) A ROUTE SPLIT block must be followed by a ROUTE JOIN block, unless the ROUTE SPLIT block is the last block in the graph.

(4) A routing slip should have an owner associated with it. Thus, in case of routing problems, the document can be sent to the owner.

By applying these rules, it is possible to ensure that the routing slip is syntactically correct. The rules clearly add restrictions on the kinds of workflows that can be designed, but are necessary in order to prevent incorrect workflows.

6.3. Permissions and security

In general, the owner who creates the routing slip can make any changes to the routing slip. The routing slip can be encrypted by the owner's private key to prevent any unauthorized changes by other nodes. Moreover, by default, other nodes will have *read-only* access to the routing slip. However, in certain situations other nodes may make changes, but only subject to constraints. These permissions must be explicitly specified by the owner by assigning appropriate values to the *rework* and *change* attributes, described in Section 4 (see Table 2).

For example, the following entry in the routing slip allows the manufacturing department to send a document back to another department, whose address appears earlier on the routing slip, to perform rework but not to change the routing slip.

lucent.com:mfg (Rework = "yes").

Furthermore, in this example the permission to change the routing slip may be necessary if the manufacturing department is not in a position to produce the order in a timely manner, and it has to be routed to a subcontractor who is not on the routing slip originally. This permission can be granted as follows with the *change* attribute:

lucent.com:mfg (Rework = "yes", Change = "yes").

After these operations, syntactic checking will be performed to ensure correctness.

6.4. Operations for combining routing slips

It is also possible to combine routing slips using the following operations:

Concatenate routing slips R1 + R2, i.e. insert R2 after R1.
Subtract routing slips R1 − R2, i.e. remove the entries in R2 from R1.
Insert routing slips, i.e. insert R2 in R1 after a certain position.
Substitute routing slips, use R2 instead of R1.

These operations can be performed only if both routing files have the same owner. Support for such operations can be provided within the DRA. These features are especially useful when the routing slips are long.

In the example above, where an order had to be sent to a subcontractor, the insert or the substitute operations would be convenient features to modify the routing slip easily. Again, after these operations, syntactic checking is necessary to ensure correctness.

6.5. Associating the routing slip with a transaction

In one sense, the tasks performed at each node that the routing slip visits might be viewed as a transaction. However, in another sense all the tasks performed by it from start to end constitute a larger transaction. This latter transaction is perhaps more meaningful because it constitutes a complete and consistent unit of work which possesses ACID properties (i.e. *atomicity*, *consistency*, *isolation* and *durability*). However, it could result in a long running inter-organizational transaction, which is not desirable. Hence, it would be helpful to split the transaction into logical subtransactions, which may run inside the larger transaction. The subtransactions would run autonomously with few constraints. Drawing such transaction boundaries appropriately requires an understanding of the semantics of each situation.

Another useful concept in the context of transactions is the notion of a *routing slip abort* operation. A node on the routing slip may abort the routing slip for a variety of reasons, and in such a case this information must be propagated to all the nodes that have seen the document. Therefore, we assume that other nodes may "undo" the routing slip, much like a transaction abort or roll back. The permission to cause such an abort has to be given explicitly by the owner using the abort attribute (see Table 2). In general, an abort can be performed at certain stages only. For instance, with a customer order, an abort can only be performed before an order is confirmed to the customer.

7. Conclusions and future work

Support for routing is a major element missing from current architectures for open electronic commerce. We described a proposal for routing documents on the Internet based on routing slips. The routing slips are a convenient metaphor for specifying flexible routing schemas and running *asynchronous* (flow-type) transactions. They can be specified in XRL, a markup language for defining a routing schema. Routing slips would be used for routing documents on the Web and implementing inter-organizational workflow systems. XRL is an extension to XML and enables efficient routing. This is an essential step towards the goal of creating truly intelligent documents on the Web, which contain semantic, routing, and permission information. XRL has a syntax similar to HTML and XML, and is easy to use and flexible.

It should be noted that XRL is a routing language and does not have a data model to accompany it. This can be viewed both as a strength and weakness. It is a strength in that it isolates the modeling of

control flows from the data modeling issues. After all, workflow modeling primarily involves modeling the control flows associated with the performance of information tasks. On the other hand, lack of a data model is a weakness in that it does not allow processing of workflows based on specific contents of the data in the documents that are being routed. Note that we do allow documents to be attached to the routing slips, but do not impose any structure or restrictions on the contents or format of these documents.

Compared to routing mechanisms in commercial workflow systems such as SAP [12], XRL offers a lightweight and more affordable alternative. A conventional workflow solution only works in a closed environment where the client and server must be able to speak a proprietary language and the system integration is usually through an Application Programming Interface (API). XRL follows the design philosophy of HTML and XML, and may be applied to production, administrative, and ad hoc workflows [7]. A detailed architecture for the document routing assistant required to support an implementation of XRL was given in Section 5.

As a next step in this research, one of the authors has embarked on a project to develop a next version of XRL, which is entirely XML compliant [23]. The advantage of doing so is to be able to take advantage of existing XML parsers to validate and parse the routing slip. Moreover, the newer version will add expressive power and overcome several shortcomings of the current version. XML is a markup language that allows users to define a set of tags, which specify the structure of a document [19,20]. For example, a user may specify tags for *name*, *age*, *department*, *email*, etc. This logical structure may be stored either in the document file itself or in an associated file called the *document type definition* (or DTD) file. While XML can help in exchange of semantic information, it still lacks routing information. Such information is critical to enable proper routing of a document within an organization and across organizations.

Finally, here is a list of future work we foresee in this area:

- Support for negotiation described in Section 2 must also be added.
- More work is required to develop a definition of *routing completeness*. How can we describe *all* routing scenarios?
- A more formal approach for defining asynchronous transactions is required.
- Implementation of the Document Routing Assistant (DRA).

References

[1] G. Alonso, C. Mohan, R. Gunthor, D. Agrawal, et al., Exotica/FMQM: a persistent message-based architecture for distributed workflow management. Information Systems Development for Decentralized Organizations, Proceedings of IFIP Working Conference on Information Systems Development for Decentralized Organizations, 1995, pp. 1–18.

[2] B.C. Arntzen, G.G. Brown, T.P. Harrison, L.L. Trafton, Global supply chain management at digital equipment corporation, Interfaces 25 (1) (1995) 69–93.

[3] M. Barbuceanu, M.S. Fox, Coordinating multiple agents in the supply chain, Proceedings of the 5th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1996 (June).

[4] M. Fan, J. Stallaert, A.B. Whinston, A web-based financial trading system, IEEE Computer 32 (4) (1999) 64–70 (April).

[5] P. Fingar, Enterprise architecture for open ecommerce, Component Strategies, SIGS Publications, 1999, pp. 44–48 (February).

[6] W. Ford, M. Baum, Secure Electronic Commerce, Prentice-Hall, Englewood Cliffs, NJ, 1997.

[7] D. Georgakopoulos, M. Hornick, A. Sheth, An overview of workflow management: from process modeling to workflow automation infrastructure. Distributed and Parallel Databases 3 (2) 1995, pp. 119–153.

[8] S. Gundavaram, CGI Programming on the Worldwide Web, O'Reilly and Associates, Sebastopol, CA, 1996.

[9] R. Glushko, J. Tenenbaum, B. Meltzer, An XML framework for agent-based E-commerce, Communications of ACM 42 (3) (1999) 106–114 (March).

[10] IEEE Computer, Special Issue on Electronic Commerce 30 (5) (1997) 44–61 (May).

[11] D. Jutla, et al., Making business sense of electronic commerce, IEEE Computer 32 (3) (1999) 67–75 (March).

[12] G. Keller, T. Teufel, SAP R/3 Process-Oriented Implementation, Addison Wesley, England, 1988.

[13] E. Krol, The Whole Internet, 2nd edn., O'Reilly and Associates, 1994.

[14] A. Kumar, J.L. Zhao, Dynamic routing and operational integrity controls in a workflow management system, Management Science 45 (2) (1999) 253–272 (February).

[15] H.L. Lee, C. Billington, The evolution of supply-chain-management models and practice at Hewlett-Packard, Interfaces 25 (5) (1995) 42–63.

[16] P. Maes, et al., Agents that buy and sell, Communications of ACM 42 (3) (1999) 81–91 (March).
[17] M. Merz, et al., Supporting electronic commerce transactions with contracting services, International Journal of Cooperative Information Systems 7 (4) (1998) 249–274 (December).
[18] A. Segev, J. Porra, M. Roldan, Internet-based EDI strategy, Decision Support Systems 21 (3) (1997) 157–170 (November).
[19] J. Sorenson, L. Wood, Document object model requirements, WWW consortium, Available at: http://www.w3.org/TR/WD-DOM/requirements.html.
[20] S. St. Laurent, XML: A Primer, MIS Press, New York, 1997.
[21] D. Strong, Decision support for exception handling and quality control in office operations, Decision Support Systems 8 (3) (1992) 217–227 (June).
[22] J. Teich, H. Wallenius, J. Wallenius, World-Wide-Web technology in support of negotiation and communication, International Journal of Technology Management 17 (1–2) (1999) 223–239.
[23] W.M.P. van der Aalst, A. Kumar, XML Based Schema Definition for Support of Inter-organizational Workflow, University of Colorado and Eindhoven University of Technology report (2000).

Akhil Kumar is currently a visiting researcher at Bell Labs, Murray Hill, NJ, on leave from University of Colorado, Boulder, where he is on the faculty of the College of Business. He holds a PhD in Information Systems from the University of California, Berkeley. In the past, he has served on the faculty at Cornell University and also worked in industry. He has published nearly 50 scholarly papers in top journals and leading international conferences in the database, and distributed and intelligent information systems areas. His current research efforts are focused in workflow systems and electronic commerce.

J. Leon Zhao is Associate Professor in the Department of Management Information Systems, University of Arizona. He holds a PhD degree in Business Administration from the Haas School of Business, University of California, Berkeley. He has also taught in Hong Kong University of Science and Technology and College of William and Mary. His current research focuses on development of workflow technologies and their applications in electronic commerce, knowledge management, and supply chain automation. His research work has appeared in such journals as *Management Science*, *Information Systems Research*, *IEEE Transactions on Knowledge and Data Engineering*, and *Journal of Management Information Systems*.