

## **XRL: An Extensible Routing Language for Electronic Commerce Applications**

Akhil Kumar<sup>1</sup>  
College of Business, Campus Box 419  
University of Colorado  
Boulder, CO 80309-0419  
Akhil.Kumar@colorado.edu

J. Leon Zhao  
School of Business and Management  
Hong Kong Univ. of Science & Technology  
Clear Water Bay, Kowloon, Hong Kong  
zhao@ust.hk

### *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 XML (Extended Markup Language) allows trading partners to exchange semantic information electronically, it does not provide support for document routing. In this paper, we propose XRL, an Extensible Routing Language that enables routing of commercial documents over the Internet and help in creating truly intelligent documents. This routing language is simple, yet powerful enough to support flexible routing of documents in the Internet environment.

### **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

---

<sup>1</sup> 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.

much in-house programming activities and are inefficient and lack interoperability. The trend is to develop more standardized architectures and techniques for electronic commerce services. An important thrust for increased productivity and interoperability is to develop more homogeneous languages for various electronic commerce activities. In this paper, we focus on the development of a workflow routing language for Internet-based electronic commerce services.

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. 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, several billion dollars worth of transactions were performed over the Web. Several architectures for Web commerce have been proposed, e.g. Commerce Net, etc. [7] and protocols like SSL and SET [4] have been developed to address security issues that arise on the Web.

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

Table 1: Stages of Web evolution

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 [11,12] makes it possible to add semantic information to a HTML document so that trading partners can understand the meaning of various fields in the document. Figure 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.
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

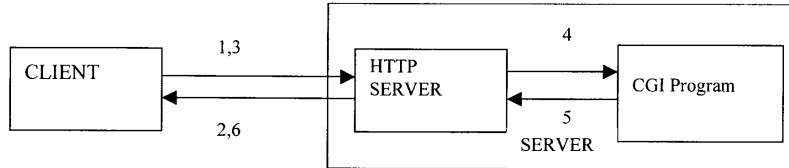


Figure 1: Existing model of the Web using CGI-gateways

server either because the underlying network is unavailable or the server is overloaded. Secondly, this is not a very efficient way of communications when a large amount of information has to be exchanged. Thirdly, this form of communication is not very conducive to *flow-type applications*, i.e. those where a flow of information/documents through multiple workers in several organizations is involved. 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], [10]). 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.

XML is a markup language that allows users to define a set of tags which specify the structure of a document [11,12]. 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.

This paper is organized as follows. Section 2 gives an overview of the intelligent document architecture based on routing slips. Next, Section 3 describes the XRL language. Then, Section 4 discusses various theoretical issues that arise in the context of this architecture and the language. Finally, Section 5 concludes the paper.

## 2. 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 which 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 of intelligent documents is most suitable for this kind of architecture.

Our general approach towards developing *intelligent documents* can be described with the notion of a *routing slip*. 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. We assume that the originator's role is to start the work on the document and specify 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, and checks the routing slip appropriately to indicate the progress made on the document. The job of the worker may be to add to, modify, comment on, or approve the document.
- 4) The routing can be simple or complex as illustrated in Figure 2. Figure 2 (a) illustrates a simple route that involves a sequential process with tasks A, B, C, D and, finally A, again, in that order. Figure 2 (b) shows a complex route that includes a parallel process and a flexible sequential process. The parallelism occurs between A and D when B and C are allowed to occur simultaneously. The flexible sequence lies in the path from D to A, where E and F can take place in any order, but not in parallel. A flexible sequential process means that some tasks can be performed in any order, but not in parallel.

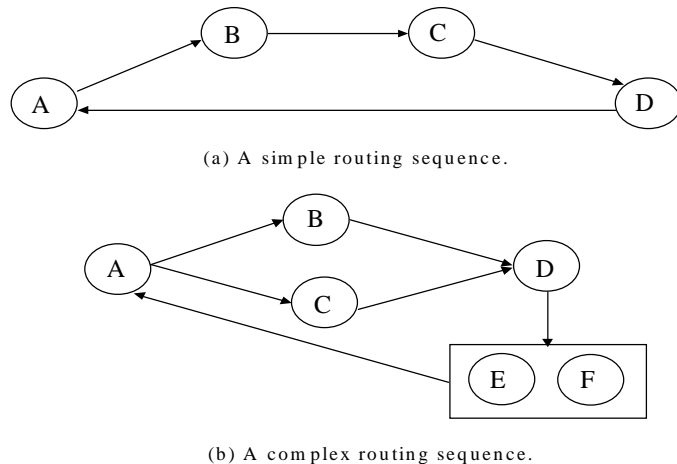


Figure 2. Simple and complex document routing.

- 5) In the case of complex routing, the worker may need some help to route the document correctly. That is, a routing facility, referred to as *document*

*routing assistant*, is needed to determine the recipients of the document based on the information in the routing slip. The document routing assistant can help the worker manage the document and the routing slip and enable the system to enforce routing constraints and trigger routing actions.

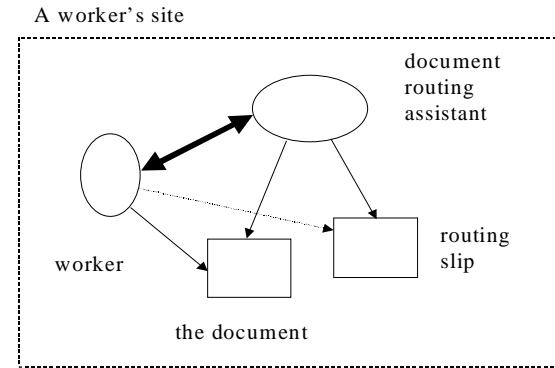


Figure 3. An illustration of document routing assistant at a worker's site.

Figure 3 illustrates the concept of document routing assistant at a worker's site. The diagram indicates that the worker can interact with the document routing assistant interactively and can read and modify the document and the routing slip when authorized. The dotted arrow from the worker and the routing slip emphasizes the authorization requirement. The routing assistant can interact with the worker and can read the document and routing slip as well. The detailed design of the document routing assistant is beyond the scope of this paper.

### 3. Overview of XRL (Extensible Routing Language)

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 [9].

### 3.1. 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 <attr1= x1, attr2=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> will be 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 HP, then to the manufacturing department, then to billing, and finally a confirmation is sent to the customer.

```
<?XRL version = 1.0 owner="hp.com:orders" ID= id1>
<ROUTE>
hp.com: orders <Attach = order.html>
hp.com: mfg
hp.com: billing
hp.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, perhaps using an HTML editor, which is available in every browser. 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.

### 3.2. Parallel Split

ROUTE SPLIT allows a document to go to any  $m$  of the  $n$  nodes in parallel ( $m < n$ ).

```
<?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>
```

### 3.3. Parallel Join

Note that a ROUTE SPLIT block is often followed by a ROUTE JOIN block. 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 next example gives an illustration.

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

```
< ROUTE JOIN 2 >
abc.com:accounts
</ ROUTE>
```

### 3.4. 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 go to *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 cannot be routed to more than one recipient at the same time. Note that unlike a SPLIT block, a SEQ block should not be followed by a JOIN block because the SEQ block always ends at a particular node.

**Example 4:** In this example, a payment invoice has to go to *any two* of the *three* vice-presidents for approval, and it can happen in *any* sequence.

```
<?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>
```

### 3.5. Nested Routing Slips

**Example 5:** In the following example, a ROUTE SPLIT block is nested inside a sequential ROUTE block. The routing pattern is that a document is sent to the accounts department and then routed in parallel to the vice-presidents and then 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 get unpredictable, and normally the application would disallow it. Note that the join is necessary only if the paths have to be merged. It is also possible that multiple copies may go along different paths and may not need to merge.

### 3.6. Routing Based on Conditions

Consider the following sequence of interactions. A customer places an order for a product. The order department checks that the product is available and confirms the order to the customer. The order is passed on then, 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="hp.com:orders">
  <ROUTE>
    hp.com: orders
      <ROUTE CONDITION <status> = "out-of-stock" >
        <SPLIT TRUE>
          compuserve.com : Akhil
```

```

    <SPLIT FALSE>
    hp.com : mfg
    hp.com : billing
    hp.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 document.

### 3.7. Exception Routing

Sometimes a need for exception or ad hoc routing can arise. In such a situation, the document must often be sent to an earlier stage. The way this would be handled is 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 subsequent to the rework stage. 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. Document history logs changes to various fields in the document. A document log is similar to a transaction log in database management systems and can be stored in a format such as <worker ID, document ID, field, old value, new value, time stamp>. Routing history contains information on the access to the document by various workers and can be recorded as <document ID, worker ID, timestamp>. 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, but does not contain information on read access. (2) A log history is more efficient to use than the document history because it is more compact.

### 3.8. 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 4.2). The *abort* attribute determines if a node may abort the routing slip (see subsequent discussion in Section 4.5).

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 <i>abort</i> the routing slip	"No"

Table 2: Various attribute names and their default values

## 4. Discussion of Various Theoretical Issues

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 if one of the addresses on the routing slips is *not valid*?

Should *cycles* be allowed in the routing slip?

### 4.1. Completeness

By completeness we mean that most common routing scenarios, that arise in production and ad hoc workflow, 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 can not 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. So, 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 usual 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.

## 4.2. 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. 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 is harder to verify and may involve sending a test email. 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 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). Drawing a directed graph with each entry as a node and the messages shown as directed links (leaving out links that correspond to confirmation messages). The presence of cycles in the graph will indicate errors.
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. It should have an owner associated with it. So 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.

## 4.3. 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 3 (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.

hp.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:

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

## 4.4. 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 the owner of the two routing files is the same. Support for such operations can be provided within a routing editor or the web browser itself. 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.

#### 4.5. 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.

However, in this context, it is useful to introduce the notion of a routing slip ABORT. 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 back out. 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.

Further study of these issues is currently under way.

### 5. Conclusions

We described a proposal for routing documents by attaching routing slips to them. The routing slips are a convenient way for specifying flexible routing schemas and running *asynchronous* (flow-type) transactions. The routing slips 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 would work as an extension to XML and enable routing to occur in an efficient manner. This is an essential step towards the goal of creating truly intelligent documents on the web, which contain both semantic, routing and permission information. The main features of XRL are that it uses a syntax similar to HTML and XML, and is easy to use and flexible.

XRL is different from XML because of several reasons:

- XML is a language that provides facilities for specifying the semantic meanings of information inside a document. It extends HTML's formatting facilities by allowing computers to interchange information. However, it does not deal with routing of document at all. XRL comes into play in Internet-based information interchange by adding the much needed routing facilities.
- XRL is based on *routing slips* that separate the hypertext and semantic information from the routing information. It leads to the concept of an *intelligent document*, which integrates semantics and routing.
- XRL helps automate business logic while XML helps automate business contents. While the goal of XML is to pack business contents into a format that can be interchanged in a flexible and open manner, the goal of XRL is to make the business process more readily executable in a flexible and open environment.

Compared to routing mechanisms in commercial workflow systems such as SAP, XRL offers a light-weight 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. XRL follows the design philosophy of HTML and XML.

From an implementation standpoint, it should be possible to define routing patterns using an editor similar to HTML or XML editors. The transport mechanism is basically SMTP (Simple Mail Transfer Protocol). Additional support will have to be provided within the browser to understand the routing information. While we have discussed the security aspects related to the integrity of the routing slip, another issue that is relevant is the security of the documents attached to the routing slip. The permissions for who can access what field in a document will have to be attached to the document itself. This information can not be included in the routing slip because documents may change and the routing slip may be attached to any document. Therefore it is helpful to uncouple such information from the routing slip. Moreover, each change made to a field by any node will need to be timestamped for audit trail purposes.

Finally, the XRL language may be applied to production, administrative and ad hoc workflows [5]. Currently we are working on approaches for defining asynchronous transactions in a formal way. We would also like to develop a

GUI authoring tool for writing and modifying XRL routing slips and implement an XRL-aware document routing algorithm.

## References

- [1] Alonso, G.; Mohan, C.; Gunthor, R.; Agrawal, D.; 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, 1-18 (1995).
- [2] Arntzen, B.C., Brown, G.G., Harrison, T.P., and Trafton, L.L. Global supply chain management at Digital Equipment Corporation. *Interfaces* vol.25, no.1, 69-93 (1995).
- [3] Barbuceanu and Fox. Coordinating multiple agents in the supply chain. *Proceedings of the 5<sup>th</sup> Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. June 1996.
- [4] Ford. W. and M. Baum, *Secure Electronic Commerce*, Prentice-Hall, (1997).
- [5] Georgakopoulos, Dimitrios, Mark Hornick, and Amit Sheth, "An overview of workflow management: from process modelling to workflow automation infrastructure", *Distributed and Parallel Databases*, vol. 3, no. 2, 119-153 (1995).
- [5] Georgakopoulos, Dimitrios, Mark Hornick, and Amit Sheth, "An overview of workflow management: from process modelling to workflow automation infrastructure", *Distributed and Parallel Databases*, Vol. 3, No. 2, pp. 119-153, 1995.
- [6] Ghundavaram, S., *CGI Programming on the Worldwide Web*, O'Reilly, (1996).
- [7] IEEE Computer, Special Issue on Electronic Commerce, vol. 30, no. 5, May (1997).
- [8] Krol, Ed, *The Whole Internet*, O'Reilly and Associates, Second ed. (1994).
- [9] Kumar, A. and Zhao, J.L. A Framework for Dynamic Routing and Operational Integrity Controls in a Workflow Management System. *Proceedings of the 29th Annual Hawaii International Conference on Systems Sciences*, (1996).
- [10] Lee, H.L. and Billington, C. The evolution of supply-chain-management models and practice at Hewlett-Packard. *Interfaces* vol.25, no.5, 42-63 (1995).
- [11] St. Laurent, Simon. *XML : A Primer*, New York : MIS: Press, (1997).
- [12] Sorenson, J. and L. Wood, "Document Object Model Requirements," WWW Consortium, <http://www.w3.org/TR/WD-DOM/requirements.html>, (1998).