A SECURE FRAMEWORK FOR MANAGING TRANSACTIONAL PROPERTIES OF BUSINESS OBJECTS

Charles A Shoniregun
School of Computing and Technology,
University of East London
Longbridge Road, Barking Campus,
Dagenham Essex,
RM8 2AS, UK.

Ziyang Duan, Subhra Bose
Reuters America Inc.
3 Times Square, 18th Floor
New York, NY 10036

Alex Logvynovskiy
Business,Computing and Information Management (BCIM),
London South Bank University
Borough Road,
SE1 0AA, UK.

Keywords: ACID, data modelling, NML, STM, web service, XML.

Abstract: A business object is a set of well-structured, persistent data associated with some predefined transactional operations. Maintaining the transactional correctness of business objects is very important, especially in financial applications. The object's correctness has to be guaranteed at any time during the lifecycle of the object. This requires that each simple operation is correct, i.e., satisfies the ACID property, and the object is in acceptable states before and after each operation. The correctness of each simple transaction can be secured and guaranteed by using a transactional database or a transaction monitor. However, the combined effect of executing a set of simple transactions may violate some business rules and leave the object in an unacceptable state. The proposed model is based on Heirarchical Statechart to specify the allowable states and transitions on a business object during its life cycle. The paper describes an XML-based framework to support application development based on this model. The framework includes an XML language for model specification, a set of tools for model definition, testing and simulation, and a set of APIs to provide business object management functionalities at runtime. The model and framework allows secure transactional properties of a business object to be defined formally and declaratively, and provides correctness guarantees at runtime. The framework facilitates fast product development and integration in a service-oriented architectural model, and provides great flexibilities for persisting data in either XML or relational databases. The experience of how to use the framework in developing a financial transactions system and the tradeoffs is based on comparison between XML and relational databases.

A Shoniregun C., Duan Z., Bose S. and Logvynovskiy A. (2005).
A SECURE FRAMEWORK FOR MANAGING TRANSACTIONAL PROPERTIES OF BUSINESS OBJECTS.
In *Proceedings of the Second International Conference on e-Business and Telecommunication Networks*, pages 57-65
DOI: 10.5220/0001407200570065

1 INTRODUCTION

Trading systems generally involve complex business logic and data transactions. The business logic reflects the business rules which might be different from one system to another depending on the target market, geographic location of the market and traders using the systems. For example, the trading rules for a stock market are different from those for a bond market, and the New York stock market has different rules from the Tokyo stock market. Business rules might change as new requirements arise from business practice. In addition, a trading system needs to accommodate many concurrent users and handle heavy volumes of transactions, each of which might involve a large amount of money. For example, billions of shares change their hands at markets such as NYSE and NASDAQ every day. Therefore, trading systems are required to be highly scalable, reliable, and with high performance guarantee. The follwoing example illustrates a typical trading system, where a trading process contains the following stages:

- *Initiation:* two traders find each other (using some search facilities) with a matched trading interest (e.g., one party wants to sell some stock shares and the other is interested in buying some shares of the same stock.) and start to contact each other. \item
- *Negotiation:* two traders negotiate on the detail of the trade until a mutual agreement is reached.
- *Settlement:* the back-office of each trader's institute confirms the deal and settles the transaction.

The trading system coordinates and monitors the whole trading process. The detail of each step is logged and the trading history can be queried afterwards. The system is integrated with other information systems to provide traders with information such as real-time market data, and integrated with participating institutions' back-office systems for transaction settlement.

The Figure 1 illustrates the example represented in a commonly used workflow notation (Hollingsworth, 1995), in which boxes represent tasks, edges represent the execution control flows, ⊕ represents *or-join*, ⊗ represents *and-split* and *and-join*.

There are several challenges in developing such a system:

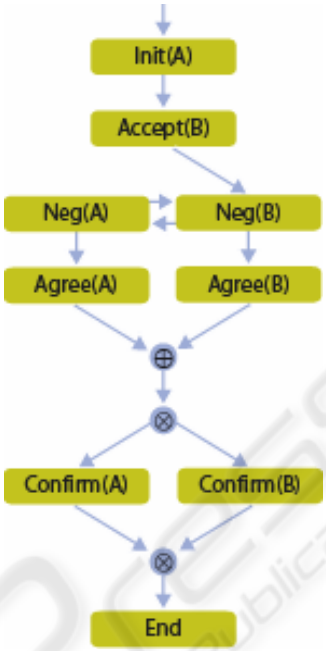

Figure 1: A negotiation workflow

- *Business rules specification*. Business rules are generally complex and can only be understood by business experts, who do not necessarily have any technical background. Even worse, no single person might have a complete picture of all business requirements. This imposes a challenging problem on system designers and developers, demanding them to spend a significant amount of time on understanding and documenting business processes before development can be started.
- *Software reusability*. With the adoption of OOD (Booch et al, 1999; Coad and Nicola, 1993) and component-based development methodologies (Buschmann et al, 1996), trading systems can be designed in a more structural way and Reusability is improved. However, since underlying trading data and business logic are usually different from one trading system to another, many components still need to be modified to reflect new data models and business requirements before they can be reused in the new system.
- *Business rules evolvement*. Conventionally, business rules are hard coded in different software components. When business rules evolve, those software components have to be updated to reflect the new requirements. To make things worse, once the business rules are dispersed and encoded into multiple software

components, they become incomprehensible and intractable, making maintenance and upgrade difficult.

- *System integration*. A trading system is often deployed in several different companies and needs to be integrated with their existing systems. Each company has its own legacy systems and regulations to do business (such as security requirements, preferred software and hardware configurations, etc.). It is important to provide an interoperable interface to facilitate system integration.

To address the above problems, an XML-based framework for trading system development was introduced. The framework provides the following:

- *A Negotiation Modelling Language (NML) is introduced to model business rules*.} NML is an XML-based language that is friendly to both business people and system developers. At the same time, it enables one to specify business rules for a trading system precisely.
- *The specification of business rules is separated from its implementation*. Specifically, business rules are specified in NML declaratively as a business process, and an engine is developed to coordinate the execution of business processes based on their NML specifications. This separation promotes software reusability and supports business rules evolvement elegantly.
- *The data model of a trading system is separated from its implementation*. The data model is specified in XML Schema, and stored in a relational database for persistence and query support. To enhance interoperability with other trading systems, FpML (Financial products Markup Language) (FpML.org, 2001), an industry standard protocol for complex financial products, is used to specify the negotiation detail of a business process. The separation of the data model from its implementation promotes software reusability since two trading systems might only differ in their data models, and one can develop a new trading system by changing only the data model specification.
- *Interoperability is facilitated in the form of web-services*. The adoption of web-services interface greatly facilitates interoperability and the integration of a trading system with information systems at different participating companies and organisations.

With the standardisation and maturity of many XML specifications and tools, it is expected to adopt more industry standards in areas such as web service orchestration and XML data query and update.

2 RELATED WORK

Workflow systems provide a way to separate the control logic from the system components, and the control logic is specified at a high level. workflows are computational models of business processes (Hollingsworth, 1995). A Workflow Management System (WFMS) provide a set of tools to specify, manage and coordinate the execution of business processes as workflows. Many workflow systems have been developed for office automation and document sharing (Schael, 1998; Mohan, 1997). During the past several years, different workflow systems have been developed, such as Exotica, ConTract and Mentor (Mohan et al, 1995; Reuter and Schwenkreis, 1995; Wodtke et al, 1996).

Though workflow models are intended to be general for all kinds of business processes including trading processes, many current workflow products are only available for special markets such as health care, telecommunication, etc. They generally are intended for end-users in a special application domain, but not for developers to build new applications in a different domain. This paper has also looked at some commercial general-purpose workflow systems, such as IBM's Flowmark, Tibco's BPM and Microsoft's Biztalk. These software shows that workflow-based integration systems are promising. However, it was found that these systems are not suitable for large-scale trading applications either because they do not meet the performance requirements or they don't provide the necessary functionalities.

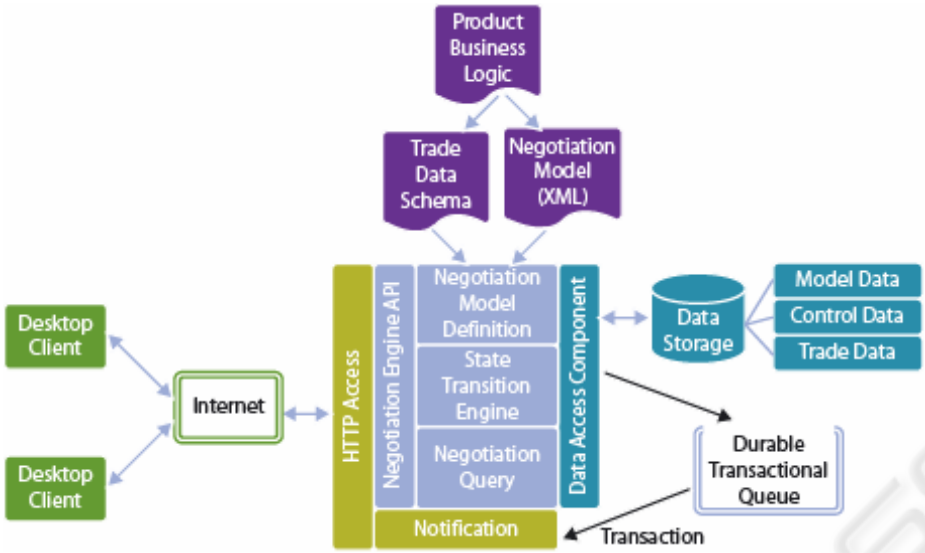

Figure 2: System architecture

A state transition model was used to specify business rules rather than using a general workflow notation. A workflow model is more powerful to model complex concurrent systems with long-lived transactions. A trading process, as a whole, can be viewed as a long-lived transaction that may last for days or months. However, each step in the process is a short transaction, and the whole process can be viewed as a sequence of state-based, event-driven short transactions. Thus, the system can be easily modelled as a state-transition machine. It has been proved that the state transition model is expressive enough for most trading systems. Trading systems usually must have high throughput. The state transition model is simpler, and an execution engine based on it can be implemented more efficiently, without the overhead to handle long-lived transactions. In addition, a state-transition model is unambiguous and easily understandable to both business and technical people.

XML has appeared as a new standard for data representation and exchange over the World-Wide-Web. XML Schema is used to specify semi-structured data types in XML. Many works have been done on understanding semi-structured data types and their relationship with relational data (Milo et al, 2000; Hosoya and Pierce, 2000; Fernandez et al, 2005). There exist several native XML databases, which do not provide the required efficiency and scalability that a trading system requires.

The Web-services based on SOAP and WSDL are widely supported in industry recently. Many standards have been proposed for web-service based business process orchestration, such as BPEL4WS (IBM, 2002) by IBM, Microsoft, etc. In the model, communications among distributed components also communicate through WSDL based web-service interfaces. However, NML mainly focuses on tracking the state transitions within a business process; whereas BPEL4WS focuses on the interface definition and service orchestration of a business process.

3 AN OVERVIEW OF THE FRAMEWORK

The negotiation engine is the main component of the application server tier. It provides the following functionalities: (1) defining and managing the negotiation models; (2) coordinating the execution of negotiation processes; (3) querying negotiation data; and (4) administrating the system. The functionalities is available to clients via a set of APIs.

An overview of this framework is presented in Figure 2, which shows three-tier architecture: the client tier, the application server tier, and the data storage tier.

The data storage tier uses a relational database at the back end. The negotiation engine retrieves and manipulates data in the data storage through the Data Access Component.

The framework provides a business logic specification language, Negotiation Modeling Language (NML). The model specification is based on state transition diagram, with additional information such as negotiation participants and their roles. A negotiation engine uses the model to

manage the negotiation process execution. A simplified example is given below:

```
<negotiation-model>
<roles>
   <role>Active</role>
   <role>Passive</role>
</roles>
<states>
   <state>Start</state>
   <state>Contacting</state>
   <stubState>Contacting</stubState>
   ...
</states>
<transitions>
   <transition>
      <name>Initiate</name>
      <allowed_source_roles>
         <role>Active</role>
      </allowed_source_roles>
      <assertion>...<assertion>
      <state_from>Start</state_from>
      <state_to>Contacting</state_to>
      <new-roles>
         ...
      </new-roles>
      <notifications>
         <notification>...<notification>
      <notifications>
      <action>...</action>
   </transition>
</transitions>
</negotiation-model>
```

In general, each negotiation model specification has the following elements:

- **Roles**: The allowed roles in a trading process. Each role has a distinct name.

```
<roles>
   <role>Talker</role>
   <role>Listener</role>
</roles>
```

- **States**: The set of states in the state transition diagram. There must be a *start* state named Start.

```
<states>
   <!-- State name -->
   <state>Start</state>
   <state>Initiating</state>
   ...
</states>
```

- **Transitions** are a set of legal transitions in a process. A simplified transition specification contains the following elements: *state_from* is the start state of a transition. *state_to* is the target state of a transition. **Assertion** is a Boolean XPath expression on the data model whose value decides if the transition is enabled or disabled. **Action** specifies a sequence of actions can be performed if a transition is successful. Such actions can be updating part of the data, or making a web service call. *Name* is the name of a transition. *Allowed_source_role* and *allowed_dest_role* are allowed roles for the source and destination party in a transition. *New_role* is the role that a participant will take after the transition. *Notification_rule* specifies the reference to an XML file that actually defines the notification rule, which is discussed later.

For example, the following XML fragment specifies a transition called Initiate, which transits the negotiation instance from the *Start* state to the *Initiation* state when invoked. In addition, it can only be invoked by a party with role Talker, and the destination party with role Listener. If the transition finishes successfully, the negotiation instance will be in *Initiation* state, and the talker and the listener switch their roles.

```
<transition>
   <name>Initiate</name>
   <state_from>Start</state_from>
   <state_to>Initiating</state_to>
   <allowed_source_role>Talker</allowed_source_rol
e>
   <allowed_dest_role>Listener</allowed_dest_role>
   <source_new_role>Talker</source_new_role>
   <dest_new_role>Listener</dest_new_role>
</transition>
```

- **Notification**: When a transition finishes, the system will notify each party the result and the current version of the payload. Notifications of different form might be sent in different situations, and clients might only accept the message formatted in a certain way. As a result, notifications are specified as a set of rule based actions. A notification rule takes the following form:

```
<notifications>
   <notification>
      <condition>...</condition>
      <prefix>...</prefix>
      <suffix>...</suffix>
      <message>...</message>
   </notification>
   <notification>...<notification>
   ...
<notifications>
```

Where *condition* is a Boolean XPath expression decides if the notification rule is enabled; *Prefix* and *Suffix* are headers and footers added on the message, and *Message* specifies the message body as an XSL transformation on the data model. The state transition machine is formally described with role assignments, assertions, and actions. For simplicity, only a flat model is described.

Definition 1. State Transition Machine (STM) is defined by a tuple as follows

$$M = (\Sigma, E, \Pi, P, A, X, \Theta_\delta, q_0, q_f),$$

where

- Σ is a finite set of **states**.
- E is a finite set of symbols called **event alphabet**.
- Π is a finite set of symbols called **participants**.
- P is a finite set of symbols called **roles**.
- $A = \{A: \Pi \rightarrow P\}$ is a set of partial functions called **role assignments**. Each $a \in A$ corresponds to a different assignment of roles to participants.
- Θ is a finite set of **actions**.
- X is a finite set of Boolean expressions called the **assertions**.
- $\delta: \Sigma \times E \times P \times X \rightarrow (\Sigma \times A \times \Theta)$ is a partial function called the **transition function**.
- $s_0 \in \Sigma$ is the **initial state**.
- $s_f \in \Sigma$ is the **final state**.

The STM works in the following fashion:

- STM is in a state s (initially, it is in the initial state s_0);
- If an event r is initiated by some participant with the role r, and if there exists c such that $\delta(s, e, r, c) = (s_1, a, q)$ and c evaluates to true, then the STM will consume the event and go into state s_1, and participants will be assigned to roles according to a;
- Otherwise, the STM will stay in the state $s_0$with the input event and role pair consumed.

The above process is repeated until the STM enters the final state s_f.

4 DATA MODELING

The assumption is that the actual payload (negotiation details) can be defined using XML schema and represented as an XML document. For example, FpML is using to describe the payload in many financial applications. The data model defines a consistent view of the negotiation data with control data and payload data. The control data works with any types of payload data.

In order to query and modify payload data efficiently, the mappping of data specified in XML schema to relational database DDL is introduced. For simplicity, the data type is modelled in the following BNF form:

```
type       := SimpleType
            | type, type
            | type*
            | type | type
            | tag[type]
SimpleType := String | Boolean | Numeric
```

The root element in the form tag[type] is shown in Figure 3. The restriction to this approach is that recursive definitions of a complex data type are not

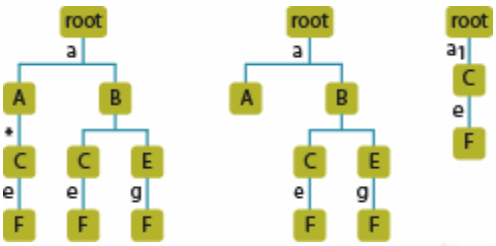

Figure 3: Root elememts

allowed to avoid unbounded depth of nested elements. The design goal of the mapping approach is to provide efficient query and update functionalities through XML, therefore the purpose is to minimise the number of generated tables and reduce the possible join operations during a query. There are some other restrictions on the details-schema, mainly features that are not supported in the current version. For example, derived types, attribute data, imported type or schema etc. The following is a valid schema.

```
R  := A, B
A  := m[C*]
B  := h[C | D]
C  := e[F]
D  := g[F]
F  := String
```

The root element is defined as a[R].

Any type definition can be represented as a finite tree with SimpleType nodes as the leaves and complex types as none-leaf nodes. Edges are annotated with element names if there is one. The tree can be construct recursively according to a simple algorithm. The detail is beyond the scope of this paper. The above example can be transformed into a tree in the following form:

The tree is transformed into a forest by cutting all the edges marked "*''. For each new tree with node N as the root, a new root element is added and the edge is annotated with the name constructed by concatenating all the edge names separated by "_'' in the root to node N path in the original tree, and an unique ID is appended at the end to avoid ambiguity.

For each tree, a table is generated in the relational database. The table name is the concatenation of the edge name between the root element and its child node. For each leaf node in the tree, a column in the table is added. The column name is also a concatenation of its ancestors as up to the element that is used in the table name. The column type is decided by its corresponding simple

data type. In addition, each table has an id column for correlation purposes.

To provide convenience and efficiency, the XML schema is extended in several ways. For example, a table name and column name can be specified explicitly instead of using the automatically generated names; An element can be designated as opaque so that its contents will be saved in a single text column as an XML string; Indexes can be defined to speed up the queries.

QueryML is an XML language designed to perform queries on negotiation data. It is based on the data model and the mapped payload schema to allow the user to create powerful queries intuitively. It can be easily converted into SQL, which can be run against its internal database.

An objective is to replace QueryML with a standard querying language such as XQuery. At present XQuery is not well supported by third party vendors yet. Current standards such as SQL or XSLT only meet part of the needs. SQL cannot be used directly because only XML-schema, but no database schema, is exposed to external applications; XSLT is a transformation language and is not suitable to be used as a general XML query language. The internal data representation is relational, therefore the SQL query from the XML query is easily constructed and efficiently executed.

A QueryML statement is defined on a given data model. The query will return a data set. Each element in the data set is of the data type defined in the data model. RootElementName is the root element of the data to be queried, and is used as the root element name of the query result. In simple, the syntax of QueryML is:

```
query     := <DataSet>
               <RootElementName>
                  statement
               </RootElementName>
             </DataSet>
statement   := <and>statements</and>
          |  <or>statements</or>
          |  <minus>statement, statement</minus>
          |  atomic
statements  := statement+
atomic      := <exp>Boolean XPath expression<exp>
```

An atomic statement denotes a query on the data set such that elements in the query result satisfies the XPath expression. The *and*, *or*, and *minus* statement represents intersection, union, and subtract of subquery result sets, respectively. The QueryML can be translated into SQL statement using and, or and subqueries easily.

In addition, an update datagram is used to specify the data image before and after the update operation to do updates on the XML data. Unlike the query statements which are performed in the relational database.

5 WEB SERVICE INTERFACE

Interfaces of the framework are accessible through the SOAP and WSDL based Web Services. For process logic control, there are two important end points: Initiate and Transition. A successful Initiate request will initiate a new process instance and notify all the participants. A successful transition request will do a transition on the process instance and notify each party of the updated state and data.

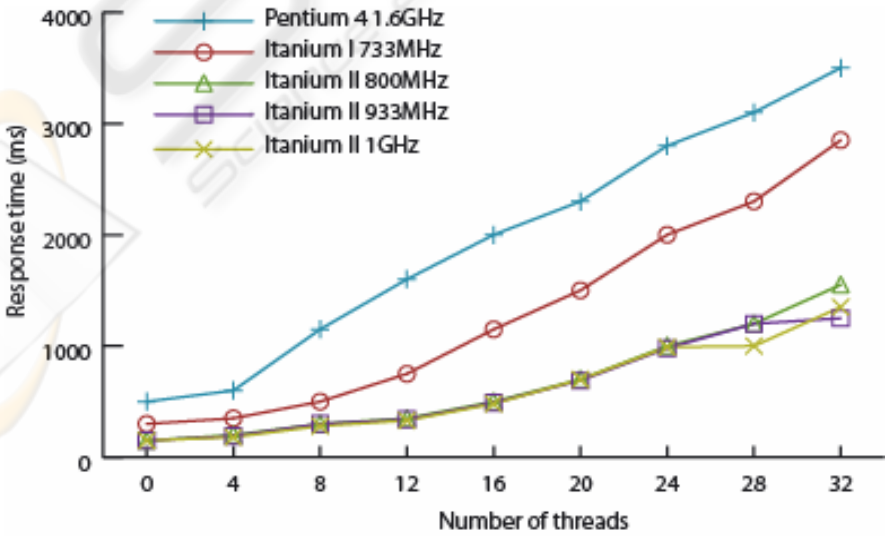

Figure 4: Performance test result

In addition, interfaces are provided to do query via the QueryML and configuration and management.

All communications among the clients are through the framework. A client send a request to server, the server validate the request, and perform database transactions, then notify all related clients of the result. Each client exposes a call back interface to the server for notification purpose.

When a transition succeeds, it can optionally make web service calls according to the model specification. This can be used to perform arbitrary tasks, such as sending an email, write a log file, or invoke another transition.

6 RESULTS AND DISCUSSION

The framework has been used in developing several mission critical trading products. It has been proved that the development cycle is reduced about one third comparing to original approaches. Man power is also saved by approximately the same amount. The reason is that the framework is now shared as a common backend for multiple products and maintained by a small group. In addition they produce a contract (the control logic and data model specification) for the product development team to follow. Each product development team does not need to worry about the backend control any more. They only need to focus on building the client interfaces. The result is a group of specialists take control of the business logic specification and execution at a high level, and another group of specialists take control of the interface design and system integration. Both groups have higher confidence about their work and better productivity (see Figure 4 for futher details). The framework was developed on the windows platform with Oracle 8i, Oracle 9i or SQL Server 2000 as the database server. The test shows the performance is excellent with a modest hardware configuration. When using the new IA64 platform, the performance can be further improved.

One benefit is that the application logic layers of several products are running the same framework. The differerence between the approaches can be found in the control logic and data model specification, which are described in several XML files. A centralised deployment of the server or a set of servers can handle many different products. Many different servers and products would have to be deployed and managed separately otherwise. With the use of web service, client software can be deployed easily across organisation boundaries.

7 CONCLUSION AND FUTURE WORK

The XML-based framework for developing trading systems was introduced. The prospects of the framework are as follows: an XML based negotiation modelling language to specify a trading process declaratively; a way to integrate data with the control logic at a high level, a mechanism to map XML-Schema data model into relational database and an XML based query language based on it; a high performance execution engine to manage and coordinate the business process; A set of predefined web service interfaces to smooth-line the integration of different clients and legacy systems and to simplify installation and deployment. The result of the research shows that the development time and cost is greatly reduced. In addition, through constant maintenance the system will be secure, but no mechanism is 100 per cent failsafe and the cost of security provision has to be weighed up against the risk for and consequence of any loss, together with the additional consideration of enabling straightforward access (Shoniregun et al, 2004).

REFERENCES

Booch, G., Rumbaugh, J. and Jacobson, I., 1999. *The Unified Modeling Language User Guide*. Addison-Wesley.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., 1996. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons.

Coad, P. and Nicola, J., 1993. *Object-Oriented Programming*. Yourdon.

Fernandez, M., Simeon, J. and Wadler, P., 2005. *An Algebra for XML Query*. Online at: http://www.cs.bell-labs.com/ wadler/topics/xml.html (Access date: January 2005).

FpML.org., 2001. *Financial products Markup Language(FpML) 1.0*. FpML.org, May. Online at: http://www.fpml.org/spec/2001/rec-fpml-1-0-2001-05-14/index.html (Access date: January 2005).

Hollingsworth, D, 1995. The workflow reference model. *Workflow Management Coalition TC00-1003*, January. Online at: http://www.wfmc.org/standards/docs/tc003v11.pdf (Access date: January 2005).

Hosoya, H. and Pierce, B., 2000. Xduce: a typed xml processing language. In *Proceedings of Third International Workshop on the Web and Databases (WebDB2000)*.

IBM, 2002. BEA Systems, and Microsoft. *Business Process Execution Language for Web Services*, Version 1.0. Online at: http://www-106.ibm.com/developerworks/library/ws-bpel/ (Access date: January 2005).

Milo, T., Suciu, D., and Vianu, V., 2000. Typechecking for XMLtransformers. In *Proceedings of the ACM Symposium on Principles of Database Systems*.

Mohan, C., Alonso, G., Gunthor, R., Kamath, M., and Reinwald, B., 1995. An overview of the exotica research project on workflow management systems. *Proc. 6th Int'l Workshop on High Performance Transaction Systems*, Asilomar, September.

Mohan, C., 1997. Recent trends in workflow management products, standards and research. In *Proc. NATO Advanced Study Institute (ASI) on Workflow Management Systems and Interoperability*, Istanbul, August.

Reuter, A. and Schwenkreis, F., 1995. Contracts - a low-level mechanism for building general purpose workflow mangement systems. *Bulletin of the Technical Committee on Data Engineering*, 18(1):4, March.

Schael, T., 1998. Workflow management systems for process organisations. *Lecture Notes in Computer Science*, 1096.

Shoniregun, C., Logvynovskiy, O., Duan, Z., Bose, S., 2004. 'Streaming and Security of Art Works on the Web'. In the *Proceedings of the IEEE Sixth International Symposium on Multimedia Software Engineering (IEEE-MSE2004)*, Miami, Florida, USA.

Wodtke, D., Weissenfels, J., Weikum, G., and Kotz Dittrich, A., 1996. The mentor project: Steps towards enterprise-wide workflow management. *Proc. 12th Int. Conf. on Data Engineering*, New Orleans, Louisiana.