An Architecture for Supply Chain Integration and Automation on the Internet [*]

IBRAHIM CINGIL AND ASUMAN DOGAC {ibrahim,asuman}@srdc.metu.edu.tr
Software Research and Development Center
Department of Computer Engineering
Middle East Technical University (METU)
06531, Ankara, Turkiye

Abstract. Electronic commerce is happening at a very fast pace and business-to-business ecommerce is taking the lead, a very important part of which is the supply chain integration and automation. There is a high demand for well accepted interoperability standards which need to be fitted together for supply chain integration to meet the business demands such as being able to integrate catalogs from different companies. This will facilitate product comparisons and producing customized catalogs. Given an anchor product anywhere on the supply chain, it should be possible to obtain information on related products that complement or add value to this anchor product. Yet another key issue is the full automation of the supply chain processes. However since a single dominant electronic commerce standard is unlikely, the supply chain integration and automation should be able to accommodate different standards like OBI or OTP. This will make it possible for users to conform to the standards of their choice.

Another important fact is that rigid supply chains can co-exist with supply chains formed on the fly where participants can transact business spontaneously since the Web is able to make the information instantly available to all trading partners. Facilitating resource discovery that is discovering information on possible partners and their catalogs on the Internet and transacting business automatically also becomes an important issue.

The architecture developed within the scope of this paper addresses these issues. We have used the emerging technologies and standards as the infrastructure of the system proposed; and integrated these to meet the needs of supply chain integration and automation and demonstrated how each of the mentioned functionality can be achieved.

1. Introduction

The electronic catalogs will much better serve the needs of the businesses today if they can achieve the following functionality through seamless interoperation of the resources on the whole supply chain of retailers, distributors and manufacturers [9]:

1. *Facilitating product comparisons and customized catalogs:* Buyers should be able to query multiple catalogs concurrently and then assemble the accumulated information in any format they prefer to be able to compare competitive products. In other words, it should be possible to integrate data from a number of different resources to create catalogs that are not only timely and information rich, but also tailored according to the customer's needs and preferences.

[*] This work is partially being supported by the Middle East Technical University, Faculty of Engineering Project No: AFP-99.03.12.02, and by the Scientific and Technical Research Council of Turkey, Project No: 197E038.

Also for large organizations that have negotiated special discounts with certain suppliers, interoperable catalog technology should make it possible to create internally distributed product listings that describe approved items and the prices set for them through master purchase agreements.

2. *Locating complementary products:* A buyer may be willing to purchase related products or a product may need several additional components before a complete system is deployed. This necessitates being able to locate compatible products in other catalogs through standardized queries. That is, once a buyer establishes a core (or "anchor") product anywhere on the supply chain, s/he should be able to readily obtain information from other catalogs describing items and services that complement or add value to this anchor product. For many suppliers this approach is also preferable to the alternative of redirecting customers to complementary catalogs since many of these catalogs may contain pointers to competitive products.

3. *Bi-directional catalog integration:* Catalog integration should be possible not only down the supply chain, that is from retailer to manufacturer but also up the supply chain that is from manufacturer to the retailer. As an example, a distributor's catalog should not only be able to obtain information about products from the original suppliers, but should also acquire information about the retail outlets where products in the catalog are offered. This will make it possible for the user to access all the information available on a particular product, regardless of where s/he has chosen to establish an anchor on the supply chain.

4. *Automation of processes on the supply chain.* Whenever a product is bought, this information should propagate down and up the supply chain automatically triggering a series of distribution, manufacturing and logistics events. As an example, the items collected in a shopping cart should automatically trigger the issuing, approval and delivery of related purchase orders electronically to the appropriate vendor organization. In response an electronic message should be sent to the buyer confirming the acceptance of the transaction, providing tracking number of the transaction and summarizing the status of the order. At the seller site, on the other hand, the related sub-processes like shipment and payment need to be automatically activated. Assuming that the buyer is a customer who contacted a retailer, it is necessary to automatically trigger the processes down the supply chain alerting necessary processes in related distributors and manufacturers.

The work described in this paper addresses these issues and is implemented within the scope of the MESChain (METU Supply Chain) project.

The technology we propose for catalog integration is to generate Extensible Markup Language (XML) documents from existing resources conforming to the Common Business Library (CBL) and also to the available industry specific standards. Given an anchor product on the supply chain, the items and services that complement this product are described using the "property" feature of Resource Description Framework (RDF). The meta data of a particular resource itself can be described through the attributes of the resource like the creator of the resource, its title, publisher, etc. There is a standardization effort in this respect, namely, Dublin Core (DC) [19] that provides for a meta data element set for describing resources. This helps not only to their discovery by search agents but also for forming the supply chains on the fly. Given an anchor product on the supply chain, bi-directional traversing on the supply chain is realized by describing up and down links through RDF properties.

XML is used in our architecture as an enabling technology since it makes it possible for business documents, forms and messages to be interoperable and comprehensible. The Electronic Data Interchange (EDI) protocol, although a success for some, has not been accepted by the majority of the business community as a way to do business electronically. Even though there have been extensive efforts to standardize EDI transactions, it remains too expensive and the software developed does not make it easy to leverage implementations across different trading partners [26]. Any solution that is targeted at enabling network economy must be one that insulates businesses operating computer systems from the daily changes that occur in a business.

In this paper, we show that a supply chain architecture can be constructed by using standard technologies without any need for specialized coding or customized programming. We demonstrate that when these technologies are used as proposed, achieving most of the required functionality reduces to executing standardized queries as demonstrated in Section 5.

Another issue addressed in the paper is the automation of supply chain processes. For this purpose catalogs are associated with catalog agents which can automatically invoke the related workflow on the supply chain for full automation of the supply chain processes. We define the workflow processes in XML conforming to a "workflow.dtd" that we provide to be replaced by the original when it becomes available through the standardization efforts currently undertaken. We also provide a component-based message-driven workflow engine in Java capable of executing the workflow definition in XML conforming to this workflow.dtd that the users of the supply chain can download and execute. The catalog agents differentiate between messages sent according to different standards and activate the related workflow process templates accordingly. This provides for the interoperability in terms of different standards.

The paper is organized as follows: Section 2 briefly describes the technologies and standards that enabled the architecture presented. To facilitate supply chain integration and automation, we propose some minor extensions to the enabling technologies. These extensions are discussed in Section 3. The architecture of the system as well as how the enabling technologies need to be fitted together are explained in Section 4. This section also contains the description of the workflow engine. In Section 5, the functionality of the system is demonstrated through

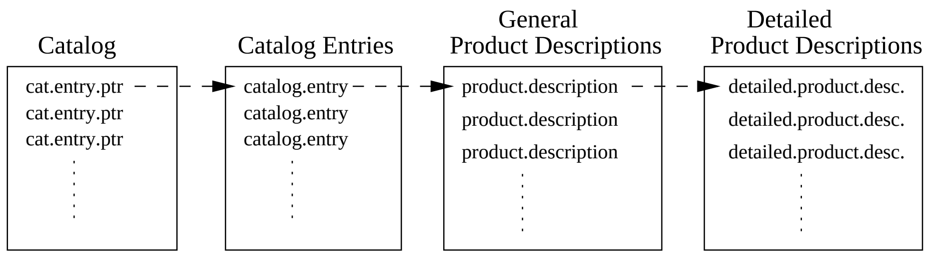

Figure 1. The electronic catalog structure in Common Business Library.

examples. The contributions of the paper are summarized in Section 6, and the related work in Section 7. Finally, Section 8 contains the conclusions.

2. Enabling Technologies and Standards

In this section, the technologies and standards that enabled the architecture presented and used in its implementation are described briefly.

2.1. Extensible Markup Language (XML) and the Common Business Library (CBL)

XML [40] has gained a great momentum and is emerging as the standard for self-describing data exchange on the Internet. Its power lies in its extensibility and ubiquity. Anyone can invent new tags for particular subject areas and define what they mean in document type definitions (DTDs). Content oriented tagging enables a computer to understand the meaning of data. But if every business uses its own XML definition for describing its data, it is not possible to achieve interoperability. The tags need to be semantically consistent across merchant boundaries. One of the efforts in this respect is the Common Business Library [11, 20, 26]. CBL consists of information models for various concepts including:

- Business forms, such as catalogs, purchase orders and invoices,
- Standard measurements, such as date, time, location and classification codes.

CBL thus provides the much needed basis to ensure interoperability among XML applications. This needs to be complemented by a set of DTDs common for specific industries, that is for vertical domains. In fact, some of the specifications for vertical domains are already available like HL7 for exchanging healthcare records, OBI (Open Buying on the Internet) [29], OTP (Open Trading Protocol) [31], and work is going on for some other domains like personal computers [34].

Related with electronic catalogs, CBL provides a standard catalog definition which includes product descriptions as well as default values for catalog operator, payment and shipment specifications. The catalog architecture of CBL is given in Figure 1.

2.2. Resource Description Framework (RDF)

RDF is a foundation for processing meta data for providing interoperability between applications that exchange machine understandable information and currently is a recommendation by the World Wide Web Consortium (W3C) [32, 33]. RDF enables meta data interoperability through the design of mechanisms that support common conventions of semantics, syntax and structure. Structure can be thought of as a formal constraint on the syntax for the consistent representation of semantics. RDF imposes the needed structural constraints to provide unambiguous methods of expressing semantics [27].

The basic data model consists of three object types: *resources* which are the things being described by RDF, *properties* which are specific aspects, attributes or relations describing a resource and *statements* that assign a value to a property of a resource. A resource can be any object that is uniquely identifiable by a Uniform Resource Identifier (URI). Values may be atomic in nature or can be other resources, which in turn may have their own properties. A collection of these properties that refers to the same resource is called a description.

Meaning in RDF is expressed through a reference to an application-specific schema which defines the terms that will be used in RDF statements and gives specific meanings to them. In other words RDF does not stipulate semantics for each resource description community, but rather provides the ability for these communities to define meta data elements as needed. Individual resource description communities define the semantics, or meaning of meta data that address their particular needs using RDF Schema Specification Language [32].

The RDF data model provides an abstract, conceptual framework; a concrete syntax is also required and XML is used for this purpose [33]. The XML namespace mechanism serves to identify RDF Schemas. The syntax and the structural constraints RDF imposes, support the consistent encoding and exchange of machine processable meta data.

As an example, the following RDF definition describes the resource "ups" as a subclass of the resource "product" and an add-on to the resource "desktop":

```
<rdf:RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
         xmlns:rds="http://www.w3.org/TR/WD-rdf-schema#"
         xmlns:MESChain="http://www.srdc.metu.edu.tr/sc/common.schema.rdf#">
 <rdf:Description ID="ups">
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-schema#Class"/>
    <rds:subClassOf resource="product"/>
    <MESChain:Add_On_to resource="desktop"/>
    <rds:label>UPS</rds:label>
    <rds:comment>All kinds of Uninterruptable Power Supplies
```

```
        that can be used for a desktop computer</rds:comment>
</rdf:Description>
```

Notice that in this example, namespace prefix "rdf" is used for qualifying RDF Syntax tags; namespace prefix "rds" is used for qualifying RDF Schema tags; and namespace prefix "MESChain" is used for qualifying the tags that we have used to define our schema. Through out the paper, the tags used in the RDF examples use these prefixes.

2.3. Querying XML and RDF Documents

There is a need for techniques and tools for extracting data from large XML documents, for translating XML data between different ontology (DTD's), for integrating XML data from multiple sources including RDF descriptions specified in XML. XML data is very similar to 'semi-structured data' [37], which has been proposed in the database research community as a data model for sources that have irregular or rapidly evolving structure.

One of the query languages available for XML is XML-QL [14] and it has been designed by applying techniques from semi-structured data. XML-QL has a SELECT-WHERE construct, like SQL, that can express queries, which extract pieces of data from XML documents, as well as transformations, which, for example, can map XML data between DTD's and can integrate XML data from different sources. Although XML-QL shares some functionality with XML's style sheet mechanism, it supports more data-intensive operations, such as joins and aggregates, and has better support for constructing new XML data, which is required by transformations. XML-QL is the main query language used in our work.

The Extensible Stylesheet Language (XSL) [42], which is still under development, also includes a transformation language. This XSL Transformations language (XSLT) [43] has recently been standardized and now is an official recommendation of the W3C. Its ability to transform data from one XML representation to another makes it an important component of any application that needs to convert the same data between different XML representations.

XSLT provides elements that define rules for how one XML document is transformed into another. The transformation is achieved by associating patterns with templates. In an XSL transformation, an XSL processor reads both an XML document and an XSL style sheet. Based on the instructions the processor finds in the XSL style sheet, it outputs a new XML document or a fragment.

The transformed XML document may use the markup and DTD of the original document or it may use a completely different set of tags. There's also special support for outputting HTML. With some effort it can also be made to output essentially arbitrary text, though it's designed primarily for XML-to-XML transformations. Since a query can be considered as a transformation from a source document to a query-result document, we use XSLT as an alternative query language in our work.

2.4. Open Buying on the Internet (OBI)

The OBI initiative [29] is automating large-scale corporate procurement of office and maintenance supplies. In the OBI architecture, a requisitioner at a Buying Organization uses a Web browser to interact with a specialized catalog at a Selling Organization. If the requisitioner places an order, the Selling Organization will transmit an order request to the Buying Organization's purchasing server for approval. If it is approved, the Buying Organization returns the order to the Selling Organization.

At an abstract level, the OBI architecture can be viewed as the interaction of four entities:

- *Requisitioner:* The requisitioner represents the end-user of the system; the person who actually places the order. The requisitioner also has a digital certificate, issued by a trusted certificate authority.

- *Buying Organization:* The buying organization represents the purchasing management and the information systems which support purchasing. These systems include an OBI server for receiving OBI Order Requests and returning OBI Orders.

- *Selling Organization:* The selling organization maintains a dynamic electronic catalog that presents accurate product and price information. This catalog can be tailored based on the organizational affiliation of the requisitioner as specified in a digital certificate.

- *Payment Authority:* Payment authorities provide authorization for the payment vehicle presented by the requisitioner. Payment authorities must provide payments to selling organizations and a timely invoice or debit to the buying organization.

2.5. Open Trading Protocol (OTP)

The Internet Open Trading Protocol [31] provides an interoperable framework for Internet commerce. It is payment system independent and it encapsulates payment systems such as SET, Mondex, CyberCash, etc. It addresses the involved parties in the trade, how it will be conducted, presentation of an offer, the method of payment, the provision of a payment receipt, the delivery of goods and the receipt of goods. The fundamental goal of the OTP effort is to produce a definition of these trading events in such a way that any two unfamiliar parties that conform to the OTP specifications will be able to complete the business safely and successfully.

A minimum useful set of OTP transactions, called "Baseline", include the following: Purchase, Refund, Value Exchange, Authentication, Withdrawal, Deposit, Payment Instrument Care, Inquiry and Ping.

The different roles that organizations can have in a trade are identified in OTP as Trading Roles, which are the following: Consumer, Merchant, Payment Handler, Delivery Handler, Merchant Customer Care Provider and Payment Instrument Customer Care Provider.

These roles may be carried out by the same organization or by different organizations. For example, in the simplest case, one physical organization (e.g. a merchant) could handle the purchase, accept the payment, deliver the goods, and provide merchant customer care. At the other extreme, a merchant can handle the purchase but can instruct the customer to pay a bank or financial institution, request that delivery be made by an overnight courier firm and to contact an organization which provides 24x7 hours of service if problems arise.

2.6. Agent Technology

An agent is a computer system situated in some environment that is capable of *autonomous action* in this environment in order to meet its design objectives. Autonomy generally means that the system acts without the direct intervention of humans (or other agents), and has control over its own actions and internal state. To better explain the autonomy of agents, consider an object 'x' in an object oriented system. The methods that object 'x' provides can be invoked by other objects and 'x' does not have any control over this. However an agent has control over its actions. Agents, in general, have the following properties:

1. They accept messages rather than being invoked by other programs.
2. They evaluate the messages that they receive and act in response rather than being told what to do, i.e., they are autonomous.
3. They discover other agents that they need to communicate.
4. They keep their state while communicating with other agents.
5. The agents can advertise the services they offer through facilitators (i.e., match makers) so that other agents can find them. To communicate with other agents in the outside world, a standard agent communication language like KQML can be used [22].

2.7. Workflow Systems

A workflow process, as defined in [39], is a coordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common business goal. A process activity is defined as a logical step or description of a piece of work that contributes towards the accomplishment of a process. A process activity may be a manual process activity and/or an automated process activity.

A workflow process is specified using a process definition language or a process definition tool, and then executed by a workflow management system (WFMS). A workflow process also defines the order of task invocation or condition(s) under which tasks must be invoked (i.e. control-flow) and data-flow between these tasks. A WFMS is a system that completely defines, manages and executes workflow processes through the execution of software whose order of execution is driven by a computer representation of the workflow process logic.

Within the scope of supply chains, workflow technology enables automation of order approval and purchasing procedures as well as assembly and/or production processes. The workflow management system developed within the scope of this work is discussed in detail in Section 4.6.

3. Integration of the Enabling Technologies

We have integrated the enabling technologies to form the proposed supply chain architecture as follows: Electronic catalog and product description data are stored in XML conforming to CBL. To provide for catalog interoperability a common DTD is necessary and CBL is one of the candidates. Another candidate like cXML [8] can also be used without a dramatic change in the proposed architecture. In MESChain architecture, meta data of electronic catalogs, the product taxonomy information, the meta data to express up and down links as well as products that add value to an anchor product are expressed in RDF. To the best of our knowledge, this is the first use of RDF in this scope and our architecture demonstrates both the feasibility and advantages of using RDF for these purposes.

A catalog agent is exploited to serve all the messages coming to a catalog. All queries against the catalogs are realized by using standard XML-QL queries via catalog agents. Standardized queries avoid any further coding effort on behalf of the participants.

All accepted customer orders and monitoring of the supply chain are handled dynamically and automatically by using the workflow technology which provides different process definitions for different purchasing protocols such as OBI, OTP, etc.

In MESChain architecture, each participant has its own catalog agent and workflow domain manager. The benefits of using a catalog agent instead of a Web server in the proposed architecture are as follows: A Web server, like an agent, can also continuously listen to a port to accept incoming messages. However in the Web server alternative, since the catalog messages need to be handled through the GET and POST methods of HTTP, a different URI or at least different parameters for each different call need to be configured on the server. Since each participant of the supply chain, like a retailer or a distributor has its own Web server, each Web server needs to be modified or customized individually. However an agent code is reusable through out the supply chain. Furthermore, when a Web server needs to communicate with an external application, for example to process supply chain queries, it uses one of the available mechanisms like CGI or Java servlets which require some coding. On the other hand, a catalog agent being specifically designed for serving catalogs, does not require any extra coding by the participants of the supply chain.

Also, a Web server accepts any request within its capability; that is, it does not have facilities to evaluate an incoming request or to keep its state. Incoming requests can be evaluated by invoking external programs and states of requests can be handled by using hidden fields on HTML pages. However both of these are dealt with indirectly. Catalog agents have built-in capabilities to evaluate incoming messages, as well as keeping the states of requests.

Automation on the proposed architecture is achieved by using the workflow technology. Workflow processes handle all the purchasing procedures, for both customer and supply orders, as well as production or assembly procedures. Using an internet based workflow system relieves any need for pre-installation of any specific software, such as CORBA for example. Furthermore specifying process definitions in XML as discussed in Section 4.6 provides for very high transportability and re-usability.

In the light of our experiences with the architecture of MESChain, there is a need in some cases for minor extensions to some of the enabling technologies to have the proposed architecture fully functional. The extensions needed are described in the following subsections.

3.1. Legacy Application Support for Catalog Interoperability

For catalog interoperability, the proposed architecture necessitates invoking some external applications from XML documents such as accessing databases, invoking workflow processes or legacy applications to dynamically modify XML documents. For example, if some of the catalog data is on a legacy database, it is necessary to invoke a wrapper program to dynamically generate catalog data to be inserted into an XML document representing the integrated catalog. Another example could be accessing an inventory database directly from an XML document to retrieve the current availability of an item in stock.

In XML, the NOTATION facility is used for invoking external applications. With this facility, it is possible to pass input parameters to the external applications, however there is no mechanism to retrieve the result of the invoked application back into XML.

With the current XML technology the following two alternatives can be used to integrate the results of an external database application into an XML document:

- If the database system is supporting XML, then it can be queried through XML-QL queries that are placed in the XML document [14].
- The XML-QL queries can be used in combination with XML "NOTATION" facility. For example, an external application that accesses the company's autonomous database may be initiated by the NOTATION facility. The external application can store its results in XML to a pre-defined location which can be accessed then by an XML-QL query.

On the other hand, the NOTATION facility can be extended to accommodate the results of an external application it initiates. An external application may generate its results in XML so that it can readily be integrated into XML data where the NOTATION facility is used. This will provide a natural mean to integrate XML with already existing applications.

Another solution that we propose is to use the XML processing application [23] syntax to interact with external applications [6, 35]. Consider the following example which retrieves the currently available quantity in stock of an item and returns the available quantity into the XML document. In this way, parsed value of the "in.stock.quantity" element will always be up-to-date although the in stock quantities reside in an external database.

```
<in.stock.quantity>
  <?EXECUTE type="PL/SQL" name="retrieve.in.stock.quantity"
    source=[SELECT available_quantity_attribute INTO avail_qty_var
            FROM   Inventory_Table
            WHERE  product_id_attribute = product_id_var]
    input=[<call.parm name="product_id_var"> "M1-CD32" </call.parm>]
    output=[<call.parm name="avail_qty_var"> in.stock.qty </call.parm>] ?>
  &in.stock.qty;
</in.stock.quantity>
```

The "avail_qty_var" and "product_id_var" are PL/SQL variables, whereas the "in.stock.qty" is an XML entity. The "M1-CD32" is the product identifier of the item in concern and is passed as a parameter to PL/SQL query from the "EXECUTE" statement and mapped to "product_id_var". The result of the query stored in "avail_qty_var" is mapped to "in.stock.qty" which is provided as the output variable and used in the XML document as an entity. Each time the "$<$ *in.stock.quantity* $>$" element is parsed, the query is executed to retrieve the current stock quantity for the product in concern.

It should be noted that the major database vendors are in an effort to fully support XML and therefore dynamic XML generation from databases will soon be readily available [30]. The advantage of our approach over such kind of a dynamic XML generation is the following: the EXECUTE instruction makes it possible to specify in the document where to get the external data and how to integrate it into the document. In this way, XML documents dynamically generated from a number of possibly heterogeneous resources can be flexibly integrated. When a change becomes necessary, for example to invoke another resource, this can be accomplished at the document level. A server program developed to do this task, on the other hand, requires the modification of the code each time a change is necessary. Furthermore, where to get the data would most likely be hard-coded in a server program.

The EXECUTE instruction as proposed can appear anywhere parsed data is allowed. The modified XML parser passes the execution to the processing application. The "type" attribute specifies the type of an application like a workflow process, a PL/SQL, an SQL or an XML-QL query. New application types can be added as needed. The "name" attribute specifies the name of the application to be executed and the "source" attribute either provides a URL or the source. The input parameters of the process, if any, can be given by the "input" attribute which is parsed and evaluated by the processing application. Parsing of the input attribute is required since the general XML entities are allowed inside the input parameters. Alternatively, the input attribute can specify a URL where the input parameters can be found. Similarly the output parameters of the process, if any, are specified by the "output" attribute which specifies a URL indicating where the output will be stored. The output attribute alternatively can give variable name specifications to store the output values. The values stored in these variables can be included into the XML document by the XML's standard general entity reference format such as: "&varname;". Note that when the result of the external application is inserted into the XML document, the document should still conform to its corresponding DTD. This is the responsibility of the person who codes the invoked external application.

There is a need to use "variables" in the definition of the "EXECUTE" processing instruction to store output values. However, variables are not readily available in XML. Variables can be accommodated into XML (with a minor change in the XML parser) by allowing the <!ENTITY ..> definitions to redefine a previously defined entity. This will enable using an entity as a variable. Alternatively, a more general variable structure can also be added to XML.

As a summary, XML is intended primarily for static data exchange on the Internet. However it may be necessary to interact with other sources from an XML document to dynamically create some XML data to be inserted into the original document. Using the EXECUTE instruction for this purpose is our contribution in this respect.

Note that there is a similar feature in HTML where it is possible to embed JavaScript in an HTML document as statements and functions within a <SCRIPT> tag, by specifying a file as the JavaScript source, or by specifying a JavaScript expression as the value of an HTML attribute [21].

3.2. Automation of Customer Orders down the Supply Chain

Full automation of customer orders down the supply chain, that is from retailer down to manufacturer, requires two mechanisms to be present in the involved parties. These are as follows:

To automate the business processes down the supply chain, automated stock control is essential in the sense that a stock danger level must be associated with each item in stock. A triggering mechanism must also be available raising a signal if the "in.stock.quantity" of a product goes below a stock danger level. This triggering mechanism enables the related workflows to be enacted to generate an automatic order to purchase or produce the missing item.

Furthermore, since each item is ordered automatically in our system, the information about the electronic catalog where the order is to be sent should also be known if the product is to be purchased. If the product is to be produced, then the production procedure of the product should be known. Note that a production process may require production or purchasing of sub parts of the product to be produced, in which case the main production process would fire required processes to produce or purchase these sub parts.

This information can either be stored in an external database, or alternatively it can be contained in the "product.description" element of CBL. In the light of the discussions above, we propose the following extensions to CBL to better accommodate the requirements of supply chain automation:

- *Inventory Control Element.* In order to automate the supply chain processes, an "inventory.control" element provided at "http://www.srdc.metu.edu.tr/sc/dtds/cbl1.2/our.extensions.mod" is introduced and added into the general product description element of CBL.

  The inventory control element contains a list of sub parts for the products that need assembling, current stock availability, stock danger level, minimum amount to order and a related workflow process pointer. When the "in.stock.quantity" decreases to the stock danger level or below an automatic supply order is generated. For a product that requires assembling, its sub parts are ordered first, then assembled and put into stock according to the specified workflow process.

- *Catalog Pointer.* A catalog pointer element is also introduced within the "inventory.control" element to point at an electronic catalog or establish links among catalogs on the supply chain. Note that catalog entry pointers in CBL do not serve this purpose and CBL does not provide catalog pointers possibly because it does not attempt to model a supply chain.

  Catalog pointers in MESChain are used for product specific purchasing purposes. For example, when the stock availability for a product is not sufficient, an automatic supply order is generated for that product and sent to a catalog agent pointed by the "catalog.pointer" contained in the "inventory.control" element. The catalog pointer differs from the "down_link" and "up_link" descriptions discussed in Section 4.4. The "up_link" and "down_link" do not provide any product specific information, but describe the roles and the participants of a supply chain.

3.3. Handling Recursive Execution in XML-QL

In an XML document there can be an element which either contains data or a pointer to this data. As an example "product.description.group" element may contain either "product.description" or "product.description. pointer" as shown in the following:

```
<!ELEMENT product.description.group (
(product.description | product.description.pointer),
 quantity.per.customer?, shipment.method.set.pointer?,
 payment.method.group.pointer?, value.monetary.group?)>
```

Note that a pointer may point to another pointer and there is no limit on the depth of this pointer chain. This implies a need for a recursive execution in some queries.

Although XSLT is capable of handling this kind of recursion, there is no support for this in XML-QL as presented in [14]. Therefore we extend the function declaration, execution and call points in XML-QL as follows:

1. The XML-QL functions should be recursive.

2. In the function declaration of XML-QL, a query block is specified in the function body which terminates with a CONSTRUCT clause. However a function can be called for some intermediate results that should not appear in the final result of the main query to conform to a particular DTD. Also the value returned by the function is not clearly specified. For this reason we introduce a RETURN clause to the function to explicitly indicate that the value returned is an intermediate result.

3. We extend the call points of a function such that a function can be called from any place where an instantiated variable can be used.

4. In XML-QL where to get the input XML document can be specified through the IN clause. However there are no means to specify where to store the result constructed. We propose INTO and APPEND_TO clauses for this purpose, right hand side of which may be a variable or a URI that represents an XML document. Note that INTO overwrites the previous content, whereas APPEND_TO appends to the previous content.

An XML-QL query processor including these extensions is available at "http://www.srdc.metu.edu.tr/ MESChain/xmlql".

3.4. Relating XML documents with their corresponding RDF descriptions

Although there are pointers from RDF descriptions to the corresponding XML documents, there is no accepted standard way of specifying the corresponding RDF description in a given XML document yet.

We consider three alternative ways of specifying RDF descriptions within XML documents:

- A special tag, say, <rdf>, can be used. In this case it is possible to associate more than one RDF description with the given XML document as shown in the following:

```
<rdf url.string="http://www.srdc.metu.edu.tr/sc/R1.catalog.rdf" />
<rdf url.string="http://www.srdc.metu.edu.tr/sc/R1.common.schema.rdf" />
```

However, defining a tag to express RDF associations necessitates a change in the DTD of the XML document. Changing the DTDs for this tag does not seem feasible. Also care should be taken in order to prevent name space collisions.

- RDF can be defined in a similar way as <!DOCUMENT ...> as shown in the following:

```
<!RDF rdf.name SYSTEM "http://www.srdc.metu.edu.tr/sc/R1.catalog.rdf" >
or
<!RDF rdf.name [ ..rdf.descriptions.. ]>
```

  This alternative does not necessitate any modifications in the related DTDs and also it is possible to accommodate more than one RDF descriptions at the cost of a minor modification in the XML parser.

- RDF link can be defined as an attribute of the XML document itself as shown in the following:

```
<?xml Version="1.0" Encoding="UTF-8"
      RDF="http://www.srdc.metu.edu.tr/sc/R1.catalog.rdf"?>
```

  Although only one RDF link can be specified in this way, this is not a serious restriction since the indicated RDF definition can include several separate RDF descriptions through XML's entity mechanism. This approach does not necessitate a change in DTD but a minor change in XML parsers is required. This seems to be the most suitable solution which is used in the architecture proposed.

4. The Architecture of the System

In the classical supply chain model, the distinctions between retailers, distributors and manufacturers are not clear cut in that manufacturers may also need to buy raw material and distributors or retailers may be assembling (i.e., producing) products. Therefore in order to come up with a more generic model we assume that all the participants on the supply chain may purchase or produce products that they sell. This removes any differences between a manufacturer and a distributor or a retailer in terms of supply chain functionality.

Figure 2 shows the proposed architecture which supports both Business-to-Business (B2B) and Business-to-Consumer (B2C) scenarios.

In the B2C scenario shown in Figure 3, a user contacts the home page of a catalog through a Web browser. Note that a search agent may help the user to locate the home pages of the relevant catalogs as explained in Section 4.1. When a user contacts the site through a browser, the HTTP server sends the home page of the catalog, on which there are facilities for querying and purchasing. When activated these facilities communicate with the catalog agent directly through XML messages conforming to "message.dtd" which is available at

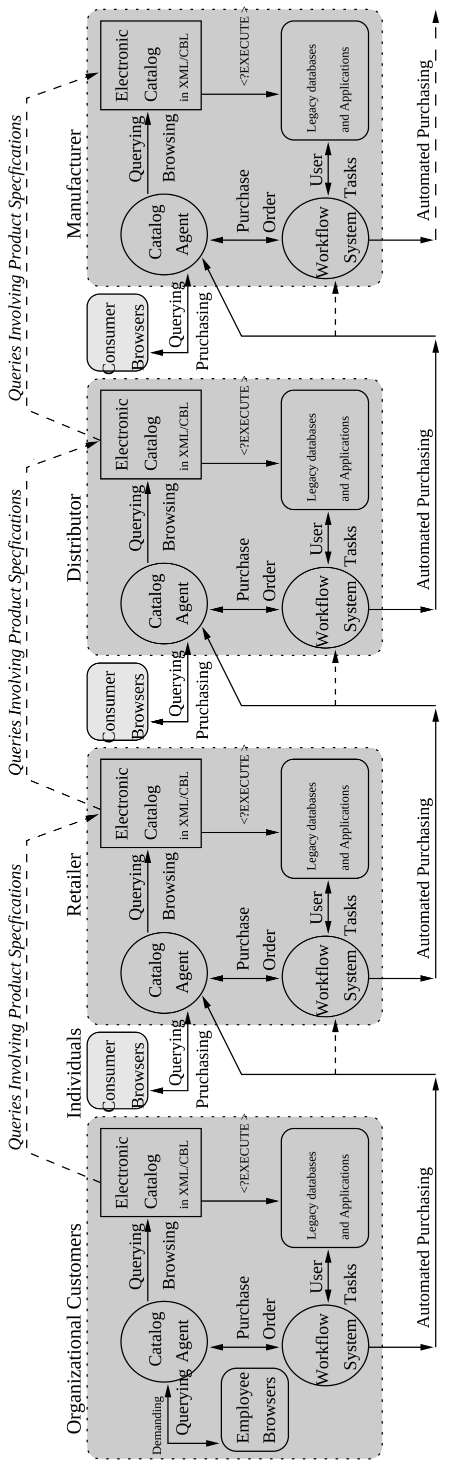

Figure 2. The proposed supply chain architecture

"http://www.srdc.metu.edu.tr/sc/dtds/message.dtd". The agent first checks an incoming message, and decides whether to accept it or not. It rejects any message which does not conform to "message.dtd", or has an undefined message type, or asks for an unauthorized functionality. If the agent decides to accept a message, it performs necessary operations according to the message type.

Message types include: "catalog/product query" messages, "purchase order" messages, "workflow process activation" messages, and "response" messages. If the message type is "query the catalog", the agent executes a standardized XML-QL query contained in the message against the catalog and product specifications in XML, and sends the result back. If the message type is purchase order, the agent then decides whether to accept the order. If it does, it determines which customer purchasing process should be activated depending on the trade protocol (like OTP, OBI, etc.) indicated in the message through the "protocol" element. Then the agent sends a process activation message to the workflow domain manager, in addition to passing necessary parameters such as the shopping basket, user identification information, delivery confirmation method and the address. The user is provided with an order tracking number, and the workflow process runs independently. The customer purchasing process, checks the stock availability from the inventory control database. If sufficient quantity is available, the process continues. Otherwise, the related task in this workflow instance sends an activation message to the workflow domain manager to initiate an automatic purchasing process down the supply chain for the missing product(s). Since there could be several types of missing products each with a different automatic purchasing process type, more than one process instance may be started. Note that each process instance started sends a purchase order message to the related catalog agent down the supply chain and this continues recursively for missing products. The customer purchasing process waits until the termination of the automatically started purchasing process(es). Once the stock becomes sufficient, the customer purchasing process proceeds. Upon successful termination of the process, a confirmation message sent to the user via her preferred media (like email, voice mail, fax, etc). We provide an example of a customer purchase process definition at "http://www.srdc.metu.edu.tr/sc/workflows/wf.processOTP.xml", realizing procurement through OTP.

In the B2B scenario shown in Figure 4, a customer organization has a customized catalog (details of which is presented in Section 4.2), and also maintains its own catalog agent. A company employee contacts the Web page of this catalog and by using the "query" and "consumables request" facilities, sends an XML message to the catalog agent. The catalog agent in response to this request activates the appropriate "Request Approval Process" depending on the employee's position in the company. The approval process checks the company inventory for the requested product(s) and if the product(s) does not exist in the company stocks and if the request is approved, an automated purchasing process(es) is started. Each process instance started sends a purchase order message to the related suppliers' catalog

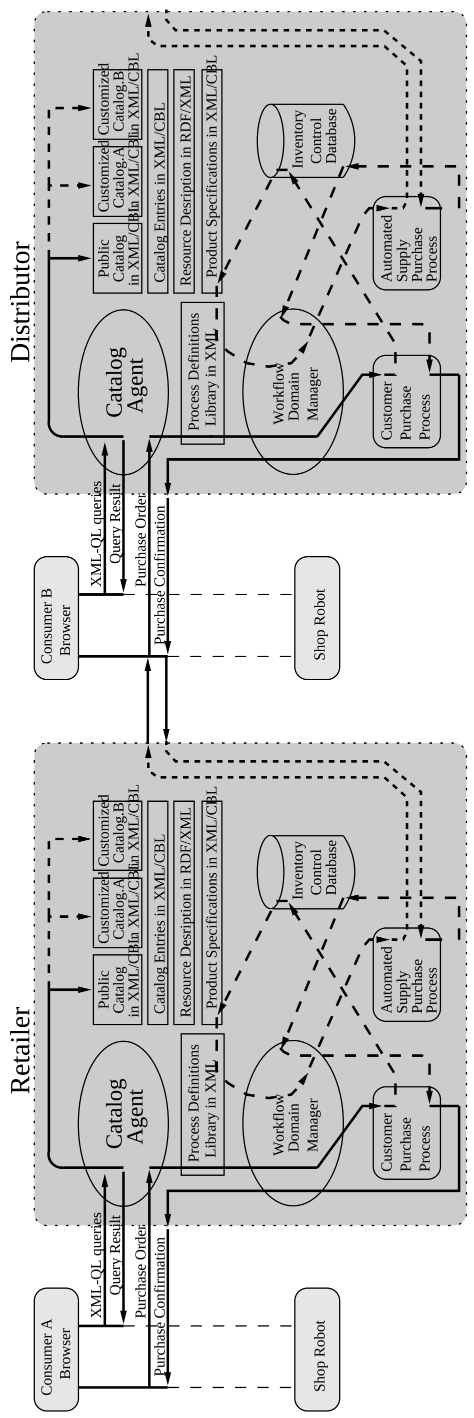
Retailer
Distributor
Consumer A Browser
Consumer B Browser
Shop Robot
Shop Robot
XML-QL queries
Query Result
Purchase Order
Purchase Confirmation
Catalog Agent
Public Catalog in XML/CBL
Customized Catalog.A in XML/CBL
Customized Catalog.B in XML/CBL
Catalog Entries in XML/CBL
Resource Desription in RDF/XML
Product Specifications in XML/CBL
Process Definitions Library in XML
Workflow Domain Manager
Inventory Control Database
Customer Purchase Process
Automated Supply Purchase Process

Figure 3. The Business-to-Consumer scenario in the proposed architecture.

agent. This may fire a recursive action down the supply chain for ordering of the missing product(s).

Note that at every site involved in the supply chain, there are RDF descriptions of product taxonomy and RDF definitions of "Added_Value" properties. These facilitate finding out the add-on products once an anchor product is established on the chain. The details of locating complementary products is provided in Sections 4.3 and 5.2. Bi-directional traversal on the supply chain is supported through "up_link" and "down_link" descriptions in RDF, details of which are given in Sections 4.4 and 5.5.

In the following a road map is presented to show how to set up such a system from an end user or a developer point of view:

1. The catalog agent, the workflow manager, the XML-QL processor code, IBM's XML4J parser (modified for the entity variables and the EXECUTE instruction) and LotusXSL processor (that uses the modified XML parser) can be downloaded from "http://www.srdc.metu.edu.tr/MESChain". These codes do not require any customization.

2. The organization should prepare its product descriptions and catalog entries in XML conforming to CBL. To facilitate this process, an empty set of product descriptions and catalog entries may be downloaded from the same address.

3. The products offered by the organization must be described in RDF according to their vertical market domain. Most likely, these RDF descriptions will be available in the future as a result of the standardization efforts similar to Dublin Core. However until then, an organization needs to develop its own RDF descriptions. Note that, this is to be done only once for a vertical domain, other organizations in the same domain may follow these RDF descriptions.

4. After preparing product descriptions, catalog entries and RDF descriptions, some minor modifications may be necessary on the standardized queries to handle different requirements of different vertical markets.

5. The organization must provide its process definitions, for example definition of a customer purchase process is the minimum requirement. Since, workflow process definitions are written in XML, they are highly transportable and common business process definitions will most likely be publicly available. Therefore, the organization should only define its own specific processes.

6. At this point, the organization is ready to join a supply chain. One more step is necessary to realize the connections to the related organizations: the "up_link" and "down_link" descriptions must be added to the RDF description of the catalog to indicate the organization's immediate suppliers and customers. Note that this necessitates the mentioned suppliers and customers to add this organization into their "up_link" and "down_link" descriptions. This is achieved by the catalog agents through the "new.partner.msg" messages as explained in Section 5.6. These links allow for up and down traversing on the supply chain.

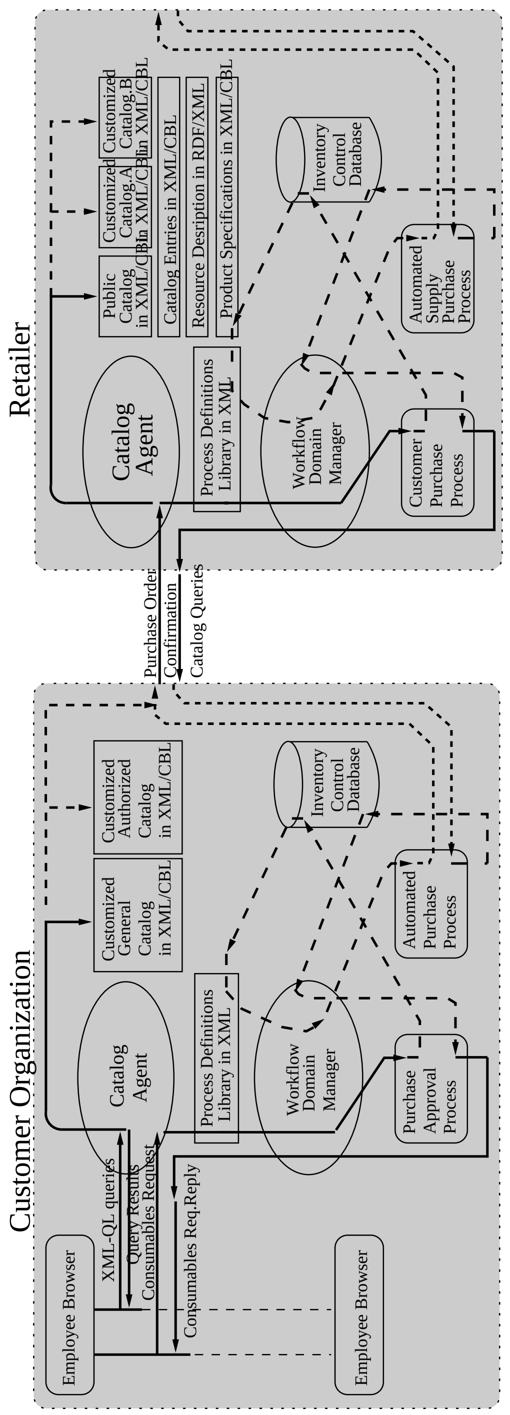
Customer Organization
Employee Browser
XML-QL queries
Query Results
Consumables Request
Consumables Req.Reply
Employee Browser
Catalog Agent
Customized General Catalog in XML/CBL
Customized Authorized Catalog in XML/CBL
Process Definitions Library in XML
Workflow Domain Manager
Inventory Control Database
Purchase Approval Process
Automated Purchase Process
Purchase Order
Confirmation
Catalog Queries
Retailer
Catalog Agent
Public Catalog in XML/CBL
Customized Catalog.A in XML/CBL
Customized Catalog.B in XML/CBL
Catalog Entries in XML/CBL
Resource Desription in RDF/XML
Product Specifications in XML/CBL
Process Definitions Library in XML
Workflow Domain Manager
Inventory Control Database
Customer Purchase Process
Automated Supply Purchase Process

Figure 4. The Business-to-Business scenario in the proposed architecture.

The following subsections describe the architecture of the system according to its functionality.

4.1. Resource discovery on the Web

The discovery of resources in our architecture is twofold: first, the resources that are electronic catalogs need to be discovered, and the second, the types of products contained in that catalog need to be discovered.

For discovery of electronic catalogs in the MESChain project, meta data of individual catalogs are expressed in RDF conforming to Dublin Core (DC) specification, an example of which can be found at "http://www.srdc.metu.edu.tr/MESChain/sc/M1.catalog.rdf".

This description makes it possible for search agents to identify the electronic catalogs and then to select catalogs that contain the desired items. It should be noted that DC, for the sake of interoperability, requires the value of the "TYPE" element to be selected from an enumerated list that is currently under development. The "electronic catalog" should be included into this list for facilitating the discovery of electronic catalogs.

We propose to specify the type of the electronic catalog, such as "manufacturer", "distributor", "retailer" and "customer", in the "SUBJECT" element. This specification facilitates on-the-fly supply chain construction since the type of an electronic catalog will also be available when the catalog is discovered by a search agent. For interoperability reasons the type values for electronic catalogs should also be selected from a pre-determined list with well-defined meanings.

In this way, not only supplier catalogs are discovered on behalf of potential buyers, but also the "customer" catalogs maintained by buyer organizations can be discovered on behalf of suppliers, for example to advertise promotions or to compete with other suppliers.

We also suggest specifying the URI of the catalog agent in the "Identifier" element, so that it will be readily available to the search agents. The search agents can then send a query message conforming to "message.dtd" to the catalog agent to verify that the page discovered is an active electronic catalog. This query may solicit further information available in the catalog.

Similarly, shop robots may also discover supplier catalogs and possibly make a database of their own. When a buyer asks a shop robot for a certain type of product, the robot may query the desired and similar products through each of the electronic catalogs listed in its database, and present the buyer intelligently combined results to facilitate comparative shopping. Note that this may require dividing a complex query into subqueries and sending each subquery to a different catalog depending on the contents of the catalog. A similar approach is discussed [24].

For discovering the types of products in a catalog, both the catalog schema expressed in CBL and the product taxonomy expressed in RDF are used. Note that the taxonomy of products offered by an organization are described in RDF according to their vertical domains in our system. CBL already provides the standards for

```
<?xml version="1.0" RDF="http://www.srdc.metu.edu.tr/sc/R1.catalog.rdf" ?>
<!DOCTYPE catalog.entries
         SYSTEM "http://www.srdc.metu.edu.tr/sc/dtds/cbl/catentry.dtd">
<catalog.entries>
......
<catalog.entry ident="R1-MB333-64">
  <product.description.group> <product.description.pointer ident="R1-MB333-64">
     <url.reference url.string="http://www.srdc.metu.edu.tr/sc/R1.proddesc.xml"/>
    </product.description.pointer>
    <value.monetary.group>
     <value.monetary.base>
         <amount.monetary currency.code="USD">450.00</amount.monetary>
     </value.monetary.base>
     <value.adjustment> <adjustment.name>Sales Tax</adjustment.name>
       <adjustment.rate>10% </adjustment.rate>
       <adjusted.monetary.value>
           <amount.monetary currency.code="USD">45.00</amount.monetary>
       </adjusted.monetary.value>
     </value.adjustment>
     <value.monetary.total>
         <amount.monetary currency.code="USD">495.00</amount.monetary>
     </value.monetary.total>
    </value.monetary.group>
  </product.description.group>
</catalog.entry>
......
</catalog.entries>
```

Figure 5. A sample "catalog.entry" element in XML/CBL

the catalog schema and most likely, standardized product taxonomy descriptions in RDF will also be available in the future as a result of the standardization efforts similar to Dublin Core.

4.2. Catalog interoperability and customized catalogs

Catalog interoperability requires handling of heterogeneous catalog data. The accepted practice to make heterogeneous data interoperable is to map it to a canonical data model. Here XML DTDs play this role, however, XML by itself is not sufficient to support interoperability; the tags need to be semantically consistent across merchant boundaries. CBL provides for the necessary standardization for horizontal domains such as for orders, invoices, catalog entries in a product independent way. Therefore, all the catalog information on the proposed supply chain architecture are either defined or transformed into XML conforming to CBL.

For organizations that keep their catalog and product information in legacy databases or applications, there is a need to convert the schema in the legacy databases to the schema of the proposed electronic catalogs and map their data

```
<product.description.group> <product.description.pointer ident="R1-MB333-64">
    <url.reference url.string="http://www.srdc.metu.edu.tr/sc/R1.proddesc.xml"/>
  </product.description.pointer>
  <value.monetary.group> <value.monetary.base>
  <amount.monetary currency.code="USD">450.00</amount.monetary>
  </value.monetary.base> <value.adjustment>
     <adjustment.name>Sales Tax</adjustment.name>
     <adjustment.rate>10% </adjustment.rate> <adjusted.monetary.value>
     <amount.monetary currency.code="USD">45.00</amount.monetary>
     </adjusted.monetary.value> </value.adjustment>
  <value.monetary.total>
     <amount.monetary currency.code="USD">495.00</amount.monetary>
  </value.monetary.total> </value.monetary.group>
</product.description.group>
```

(a) A sample output string produced by a "wrapper" program.

```
<?xml version="1.0" RDF="http://www.srdc.metu.edu.tr/sc/R1.catalog.rdf" ?>
<!DOCTYPE catalog.entries
          SYSTEM "http://www.srdc.metu.edu.tr/sc/dtds/cbl/catentry.dtd">
<catalog.entries>
......
<catalog.entry ident="R1-MB333-64">
  <?EXECUTE type="Unix_Shell"  name="Get.CatEntry"
       source="http://www.srdc.metu.edu.tr/sc/wrappers/Get.Catentry"
       input=[<call.parm name="product.id">R1-MB333-64</call.parm>]
       output=[<call.parm name="cat.entry.content"> centry </call.parm>] ?>
  &centry;
</catalog.entry>
......
</catalog.entries>
```

(b) An example coding of EXECUTE instruction for the wrapper.

Figure 6. Using a wrapper program to dynamically generate contents of a "catalog.entry".

into XML conforming to proper catalog or product DTDs of CBL. The schema conversion problem is not an easy task, but it is unavoidable for interoperability. Schema conversion problem is out of the scope of this paper, since it is application dependent. However, it should be noted that once this schema conversion process is defined, it can be directly embedded into wrapper programs. For schema integration we propose to use the EXECUTE instruction introduced in Section 3.1. Assume the sample catalog entry given in Figure 5. The EXECUTE instruction in Figure 6 (b) invokes the wrapper program of the related legacy application to produce the string in Figure 6 (a). In this way, the necessary XML document is both dynamically and automatically generated.

The efforts are also continuing for producing product specific standards for vertical industry domains such as for computer industry. In MESChain, the class hierarchy of the products are defined through RDF descriptions and the industry specific DTDs are used to describe the products in the leaf classes. For example, as shown in Figure 11, the "computer" class has two subclasses as "laptop" and "desktop". The "laptop.dtd" has already been produced by the RosettaNet project and we provide the remaining DTDs like "desktop.dtd".

As demonstrated in Section 5.1, catalog interoperability can be achieved through standardized queries once the catalog and product DTDs of CBL are used as the canonical model.

The customized catalogs, on the other hand, can be generated in the following ways:

- A participant can generate catalogs specific to its certain customers with special discount information. But this may require the repetition of some catalog entries according to CBL's catalog entry DTD. Yet in this way it is possible to give different products in different catalogs. As an example, a supplier may wish to provide its latest products only to a specified set of customers. Digital certificates are used by the catalog agent for the identification of customers.

- A customer can maintain its own catalog which may contain entries from different supplier catalogs. It is possible to achieve this through standardized queries since all the catalogs are generated in XML conforming to standard DTDs. In this way, the catalogs not only contain timely information like the most recent promotions but also will be tailored according to the customer's needs and preferences. The customer catalog will contain only the products that the customer is interested in and in the style s/he prefers through XML's style sheets. Note that a customer organization may also maintain more than one catalog, for example different catalogs for different departments, or different catalogs for different employee positions. The catalog agent of the customer organization, may select appropriate catalog according to the digital certificate identifying the employee. Producing customized catalogs for the consumer through standardized queries is depicted through an example in Section 5.1.

4.3. Discovery of items and services that add value

On the supply chain it is also necessary to provide information about the items or services that complement or add value to a product. Note that, a product taxonomy is also necessary in identifying this information since the add on products of a super class (e.g. computer) can also be the add on products of its sub class (e.g. desktop). The following three alternatives are considered for representing the add-on product data in the electronic catalogs:

1. *Keeping the add-on product data in the product definitions:* This causes redundancy since the same add on product must be specified repetitively for each instance of a product. For example, assuming that "UPSs are add-on products to desktops", each instance of a desktop product will contain an "add-on" tag to specify "ups" as shown in Figure 7 (a). When a new add-on product is in-

```
<product ident="d1"> <taxon>desktop</taxon> <addon>ups</addon> ...... </product>
<product ident="d2"> <taxon>desktop</taxon> <addon>ups</addon> ...... </product>
<product ident="d3"> <taxon>desktop</taxon> <addon>ups</addon> ...... </product>
<product ident="d4"> <taxon>desktop</taxon> <addon>ups</addon> ...... </product>
(a) Add-On product information kept as a part of the product descriptions.

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE catalog [<!ELEMENT taxonomy ( taxon+ )>
   <!ELEMENT taxon       (name, added.value*, addon.to*, part.req*, part.opt*,
                          parent*, child*) >
   <!ELEMENT name        (#PCDATA) >
   <!ELEMENT added.value (#PCDATA) >
   <!ELEMENT addon.to    (#PCDATA) >
   <!ELEMENT part.req    (#PCDATA) >
   <!ELEMENT part.opt    (#PCDATA) >
   <!ELEMENT parent      (#PCDATA) >
   <!ELEMENT child       (#PCDATA) > ]>
<taxonomy>
<taxon><name>computer</name><added.value>printer</added.value>
   <parent>product</product><child>desktop</child><child>laptop</child></taxon>
<taxon><name>desktop</name><added.value>ups</addedd.value>
  <part.req>monitor</part.req><part.req>keyboard</part.req>
  <part.opt>cdrom</part.opt><parent>computer</product></taxon>
<taxon><name>ups</name><addon.to>desktop</addon.to>
  <parent>product</product></taxon>
</taxonomy>
(b) A sample product taxonomy definition.
```

Figure 7. Three alternatives for representing the "added value" information for products.

troduced to desktops, all these instances should be updated to reflect this new add-on product.

2. *Using a "taxonomy.dtd":* Assume the sample taxonomy DTD given in Figure 7 (b) which describes a product taxonomy. This "taxonomy.dtd" specifies the class/subclass hierarchy through "parent" and "child" elements, and the added-value property through the "added.value" element.

3. *Using RDF definitions:* Both the product taxonomy and the "Added_Value" property can be expressed by using RDF Schema Definition Language, more specifically through "subClassOf" and "Property" features of RDF. Figure 8 shows the RDF definition of the "Added_Value" property and definition of the resources "desktop" and "ups". The class/subclass hierarchy and the added value property definition are explicitly given, and the "domain" and "range" constraints indicate the type of resources that the property applies.

   Note that RDF, by giving a description for each resource individually, like "desktop" and "ups" in the example, declares what kind of properties these resources may have.

```
<rdf:Description ID="Added_Value">
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-syntax#Property"/>
    <rds:label>Add On Product</rds:label>
    <rds:domain resource="Product">
    <rds:range  resource="Product">
    <rds:comment>The Added_Value property indicates the possible
         Add_On products for an anchor product</rds:comment>
</rdf:Description>
<rdf:Description ID="desktop">
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-schema#Class"/>
    <rds:subClassOf resource="computer"/>
    <MESChain:Added_Value resource="ups"/>
    <MESChain:Part_Required resource="monitor"/>
    <MESChain:Part_Required resource="keyboard"/>
    <MESChain:Part_Optional resource="cdrom"/>
    <rds:label>Computer</rds:label>
    <rds:comment>All kinds of desktop computers</rds:comment>
</rdf:Description>
<rdf:Description ID="ups">
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-schema#Class"/>
    <rds:subClassOf resource="Product"/>
    <MESChain:Add_On_to resource="desktop"/>
    <rds:label>UPS</rds:label>
    <rds:comment>All kinds of Uninterruptable
         Power Supplies for desktops</rds:comment>
</rdf:Description>
```

Figure 8. RDF description and use of the "Added_Value" property.

The XML DTDs in comparison to RDF descriptions suffer from the following disadvantages when it comes to expressing relationships among resources:

- "subClassOf" being a part of RDF Schema Definition Language has a well defined meaning; whereas the tags like "parent" or "child" need interpretation.
- The properties that resources may have, are not explicit in the XML DTDs, since all the properties of the resources need to conform to the same DTD. In the example given, "ups" does not have any "add-on" products and this is explicit from the RDF description in Figure 8; however this fact is not apparent in the "taxonomy.dtd" in Figure 7 (b), since all the products conform to the same DTD.
- Furthermore, "domain" and "range" constraints in RDF provide for their proper use through automatic validation of constraints on the resources that they apply. DTDs do not provide for this.

If we summarize, the first approach presented does not differentiate data from meta data and RDF descriptions have better expressive power compared to XML DTDs in expressing relationships.

In MESChain we used the third alternative where the "subClassOf" and "property" features of RDF are used to express the information about the items or services that complement or add value to a product. In the descriptions given in Figure 8, the "property" is defined in a generic way, that is, UPSs are defined as "Add-On" products for desktop computers without giving any specific instance information, since we assumed that all the instances of UPSs are add on products for all the desktop computers.

Finding related or add-on products for a given anchor product from the RDF descriptions through standard queries is depicted by an example in Section 5.2.

4.4. Bi-directional traversal on the supply chain

Generally a participant on a supply chain knows about its suppliers and buyers because they have some kind of pre-established agreements among them. For example, a distributor knows which products it buys from which manufacturer or a retailer knows which products it buys from which distributor and the manufacturer of these products. However, the information in the reverse direction is generally not available.

We have defined the relationships among the catalogs on the supply chain, in other words, participant roles through RDF property. At any point on the supply chain, all the immediate suppliers to this point are defined as "down_link" and all the immediate purchasers of this point are defined as "up_link" providing the capability for bi-directional traversals on the chain. RDF definitions of these links can be found at "http://www.srdc.metu.edu.tr/MESChain/sc/common.schema.rdf".

Instance specific values for these properties need to be specified as RDF statements as shown in the following example which describes that distributor D1 has two manufacturers (M1,M2) down the chain and two retailers (R1,R2) up the chain:

```
<rdf:Description about="http://www.srdc.metu.edu.tr/sc/D1.catalog.xml">
<MESChain:down_link resource="http://www.srdc.metu.edu.tr/sc/M1.catalog.xml"/>
<MESChain:down_link resource="http://www.srdc.metu.edu.tr/sc/M2.catalog.xml"/>
<MESChain:up_link   resource="http://www.srdc.metu.edu.tr/sc/R1.catalog.xml"/>
<MESChain:up_link   resource="http://www.srdc.metu.edu.tr/sc/R2.catalog.xml"/>
</rdf:Description>
```

These links are created automatically by the catalog agents through the new partner messages as discussed in Section 5.6.

An example is given in Section 5.5 that illustrates an upward traversal of the supply chain from a manufacturer's catalog to a retailer's catalog through a standardized query.

4.5. The architecture of catalog agents

In MESChain each catalog is associated with a catalog agent. The catalog agent is used in querying catalogs; in automating the supply chain by automatically invoking the related workflow processes and also for "on-the-fly supply chain construction" as described in Section 5.6.

Using a catalog agent instead of a Web server provides the following benefits: A catalog agent, being specifically designed for serving electronic catalogs, does not require any modification or customization of the agent code. A catalog agent also keeps internal state while communicating with involved parties and has autonomy and specialized knowledge on how to handle incoming messages.

The catalog agent handles incoming messages according to the "message.dtd" which contains the following message types:

- Query Catalog ("query.catalog.msg") messages: The standardized XML-QL query contained in the message is executed against the catalog and the product specifications in XML, and the result is sent back.

- Purchase Order ("purchase.msg") messages: If the agent decides to accept the purchase order, it uses the "protocol" element in the message to decide on which trade protocol (like OTP, OBI, etc.) to use. It is possible to accommodate new standards by introducing new workflow templates and by informing the catalog agents about the new protocols and the related workflow templates.

- Workflow Process Activation ("wf.process.run") messages: The agent sends this message to the domain manager to start a process instance.

- Order Tracking ("order.tracking.msg") message: A customer sends this message to the catalog agent through the facilities on the home page of the catalog with the proper "order tracking id".

- Workflow Process Status Querying ("wf.query") messages: The agent sends this message to the domain manager when it receives an order tracking message from a customer.

- Partnership Negotiation ("negotiate.msg") message: This message is used by the catalog agent to handle the pre-negotiation to form a new partnership which needs to be approved by an approval process.

- New Partner ("new.partner.msg") message: When a partnership is approved, the catalog agent sends this message to the related catalog agent to establish the "up_link" and "down_link" descriptions on the supply chain.

- Response ("response.msg") messages: In response to every message a response message is sent.

The catalog agents have the following properties:

- They are autonomous; they need to listen to a port for incoming messages.

- They handle different message types and act accordingly.

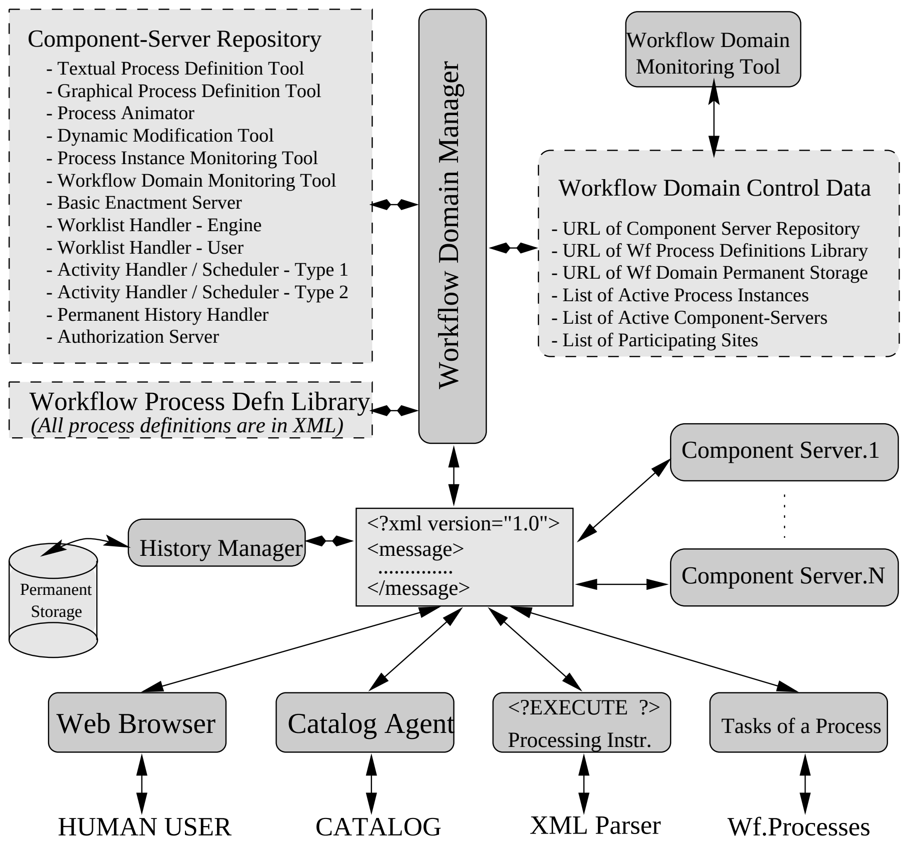
Component-Server Repository
- Textual Process Definition Tool
- Graphical Process Definition Tool
- Process Animator
- Dynamic Modification Tool
- Process Instance Monitoring Tool
- Workflow Domain Monitoring Tool
- Basic Enactment Server
- Worklist Handler - Engine
- Worklist Handler - User
- Activity Handler / Scheduler - Type 1
- Activity Handler / Scheduler - Type 2
- Permanent History Handler
- Authorization Server
Workflow Domain Manager
Workflow Domain Monitoring Tool
Workflow Domain Control Data
- URL of Component Server Repository
- URL of Wf Process Definitions Library
- URL of Wf Domain Permanent Storage
- List of Active Process Instances
- List of Active Component-Servers
- List of Participating Sites
Workflow Process Defn Library
(All process definitions are in XML)
Component Server.1
Component Server.N
History Manager
Permanent Storage
<?xml version="1.0">
<message>
....
</message>
Web Browser
Catalog Agent
<?EXECUTE ?> Processing Instr.
Tasks of a Process
HUMAN USER
CATALOG
XML Parser
Wf.Processes

Figure 9. The proposed message-driven, component-based and adaptive workflow system architecture.

- They preserve their state while communicating with other agents and with the domain manager while processing messages.
- They communicate with other catalog agents for establishing new partnership relations automatically.

MESChain Agents communicate in XML over TCP/IP.

4.6. The workflow architecture

A workflow system is provided to fully automate the supply chain processes in co-operation with the catalog agent. Workflow processes are defined in XML conforming to a "workflow.dtd" which is provided in "http://www.srdc.metu.edu.tr/sc/dtds

/workflow.dtd". Using workflow systems in supply chain integration is an accepted practice. However, the workflow architecture proposed in this paper better fits to supply chain automation. The advantages of this architecture are discussed in Section 6.

The architecture of the workflow system is message driven and component based as shown in Figure 9. An implementation is available in [44]. Each participant on the supply chain has a Workflow Domain Manager which runs in close contact with the catalog agent. The Workflow Domain Manager and other components of the workflow system are implemented as Java objects and they communicate with each other through XML messages.

The basic components of the workflow system are presented in the following:

1. Workflow Domain Manager: The domain manager is the server of the system. It communicates via XML messages conforming to "message.dtd", and it can perform any of its functionality through these messages. For example, a catalog agent sends only an XML message ("wf.process.run") to initiate the related process when a customer purchase order is accepted. The domain manager, in response to a "wf.query" message provides the monitoring information to the catalog agent.

   Human clients access the domain manager through a Web browser for administrative purposes like defining new processes or monitoring the system. The domain manager downloads appropriate Java applets to the client which then handle subsequent requests of the same client for that particular service which is provided by a component server. The domain manager keeps runtime information such as list of active process instances, active component servers, list of participating sites, etc. for domain monitoring purposes.

2. Workflow Process Object (WPO): When the catalog agent or an authorized user wants to initiate a new instance of a pre-specified workflow process, the Domain Manager creates a new "Workflow Process Object". The main method of this object is the "Basic Enactment Method" which is activated by the Domain Manager on behalf of the client. The WPO contains all the data (such as workflow process definition, workflow relevant data, enactment history of that instance up to the current execution point, etc.) required to complete the execution of the process instance, or to migrate the process instance from one site to another, or to rescue an instance in case of failures. Since a WPO contains its own copy of the workflow process definition and all the run-time information about its own process instance, the dynamic modification of the workflow process definition on instance basis is simply enabled by dynamically modifying the WPO.

   WPOs exist only for active process instances. When a process instance terminates, its WPO is destroyed after its history is permanently saved by the History Manager.

3. Component-Server Repository: The components of the system are implemented as Java objects and are activated by the domain manager as requested by the executing process instances. The human interaction components like Dynamic Modification tool, on the other hand, are accessed by the authorized users through Java applets. The Component-Server Repository includes the following components for human interaction:

    - Workflow Process Definition Tool: Authorized users are allowed to define new workflow processes or to delete previously defined processes. The process definition is syntactically verified and permanently stored in the Workflow Process Definition Library in XML.
    - Dynamic Modification Tool: Authorized users are allowed to modify a particular workflow process instance at run time to respond to external changes that cause variations in the pre-specified process definition. In such a case, the modifications can be applied to executing instances selectively or to all instances of the same workflow process if required. The modifications can also be reflected to the template definitions in the Workflow Process Definition Library if needed.
    - Process Instance Monitoring Tool: Users are allowed to trace workflow process instances they have initiated and extract run-time information about the current execution status of an instance. Collecting and measuring process enactment data are needed to improve subsequent process enactment as well as documenting what process actions actually occurred in what order. This feature provides data for optimization and evaluation of process definitions. Note that an authorized user can monitor any process instance.

4. Workflow Process Definitions Library: Workflow definitions (i.e., the process templates), organizational role definitions, and participant-role assignments are durably stored into this library. Only workflow process definition tool and dynamic modification tool may insert or update workflow process templates in this library. However, a new process definition may remotely be inserted into the library through an XML message sent to the domain manager provided that the message contains the process definition.

5. History Manager: The History Manager handles the database that stores the information about workflow process instances which have been enacted to completion to provide history related information to its clients (e.g. for data mining purposes). It should be noted that the history of active process instances are stored in the WPO itself.

The implementation of this workflow architecture provided in [44] requires only TCP/IP connection to communicate with its components, the catalog agent and domain managers of other organizations. Since the workflow system is message based, it does not need any special software such as CORBA to remotely execute tasks.

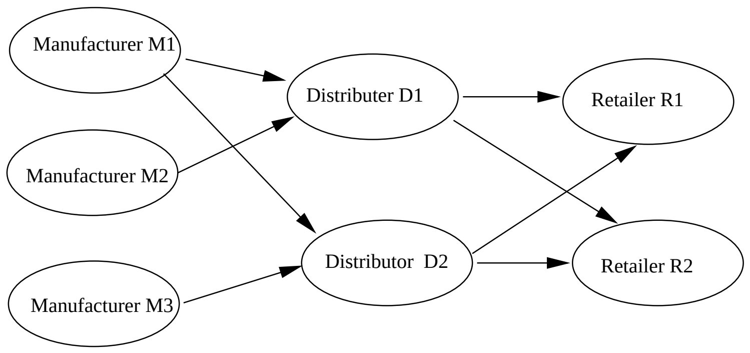

Figure 10. An Example Supply Chain

5. The Functionality and Implementation Status of the Proposed Architecture

A proof of concept prototype of the proposed system developed within the scope of the MESChain (METU Supply Chain) project is available from "http://www.srdc.metu.edu.tr/ MESChain". The prototype is implemented in Java JDK 1.1.5. IBM's XML4J parser [41] and LotusXSL 1.0.1 [25] are used to parse and transform XML documents. However we have modified the XML parser for the entity variables and the EXECUTE instruction as discussed in Section 3.1, and arranged the XSLT processor to use this modified XML parser. We have implemented the XML-QL processor by using Java and JavaCC. The Web server used is Apache 1.39, and the browser is Netscape 4.5. The system runs on Sun Ultra-10 operating under Sun OS 5.6. Since Netscape 4.5 does not support XML documents, all the XML results obtained are converted to HTML through XSLT.

Figure 10 shows a sample supply chain where all the involved catalogs are expressed in XML conforming to CBL catalog architecture given in Figure 1. Figure 11 presents a simplified example of the product taxonomy for the personal computer industry.

In the following subsections the functionality of the proposed architecture shown in Figure 2 is described through detailed examples which are based on the sample supply chain and product taxonomy given in Figures 10 and 11. All the queries (both in XML-QL and XSLT), query results, sample catalog data and sample product taxonomy descriptions mentioned in the examples are available at "http:// www.srdc.metu.edu.tr/ MESChain/sc".

5.1. Producing customized catalog for the consumer

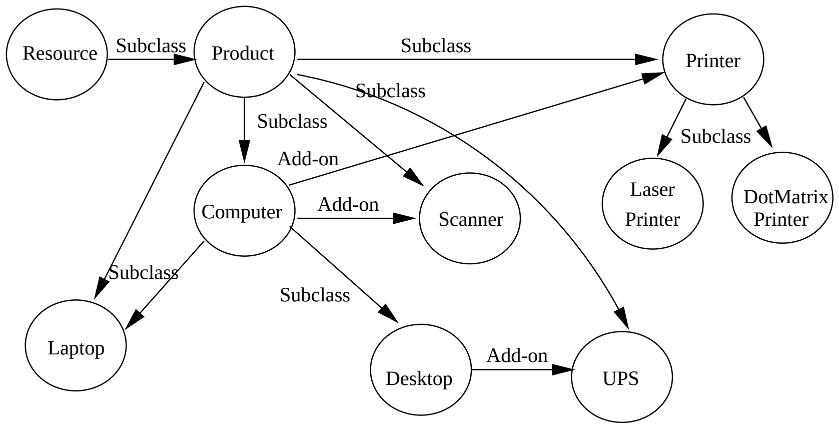

Figure 11. RDF Schema Description of the Example Resource

Assume that an individual customer C1 contacts retailer R1's catalog to buy a 300 MHz or faster Pentium PC. The standardized query given in Figure 12 will return the items in the retailer's catalog that satisfies the customer's request.

The first part of the query in Figure 12.b matches each catalog entry pointer found in the catalog of the retailer R1. The second part of the query accesses corresponding catalog entries located in "catentry_url" and identified by "cei". Each accessed catalog entry points to a product description element located in "proddesc_url" and identified by "pdi". The third part of the query calls the function "Get.Proddesc.Element" to find the "product.description.general" element and selects the product only if it contains a feature "Clock Speed" with a value greater or equal to 300. Here the function contains two sub queries, such that the second one is executed only if the first one fails, because the "product.description" element will either contain a "product.description.general" element or a pointer to it. The first sub query returns the "product.description.general" element if it is present in the "product.description". The second sub query on the other hand, follows the "product.description.general.pointer" link recursively until "product.description.general" element is found. Assume that the catalog entry pointers of the following items are returned as the result of this query:
R1-PC333-A, R1-PC333-B.

5.2. Finding related or add-on products for a given anchor product

Assume that the user wishes to find out about add on products as well. In order to do this, resource type of the selected products are obtained from the product

```
FUNCTION Get.Proddesc.General.Element (in $pdi, in $proddesc_url )
 { WHERE <product.description ident = "$pdi">
           <product.description.general> </> ELEMENT_AS $pdge
         </> IN $proddesc_url
   RETURN $pdge }
 { WHERE <product.description ident = "$pdi">
           <product.description.general.pointer ident = "$pdgi">
           <url.reference url.string = "$pdgidesc_url">
         </></></> IN $proddesc_url
   RETURN Get.Proddesc.General.Element($pdgi, $pdgidesc_url) }
END
               (a)

WHERE <catalog>
       <catalog.entry.pointer ident = "$cei">
        <url.reference url.string = "$catentry_url">
       </></> ELEMENT_AS $cep_element
      </> IN "www.srdc.metu.edu.tr/sc/R1.catalog.xml",

      <catalog.entry ident = "$cei">
       <product.description.group>
        <product.description.pointer ident = "$pdi">
         <url.reference url.string = "$proddesc_url">
      </></></></> IN "$catentry_url",

      <product.description.general>
       <keyword.set><keyword>Desktop</></>
        <feature.set> <feature.group>
         <feature.name>Clock Speed</>
         <feature.name.value><mhz>$mhz_value</></>
      </></></> IN Get.Proddesc.General.Element($pdi, $proddesc_url),

      EXPR "($mhz_value >= 300)"

CONSTRUCT $cep_element INTO "result1.xml"

               (b)
```

Figure 12. A standardized XML query to find out PCs with a 300 Mhz or faster CPU

description. Then, the RDF description of each of these resources are searched to obtain a list of resource types of add-on products specified by the "Added-Value" property. Note that "computer" is the super class of "desktop" and the add on products of "computer" are also the add on products of "desktop". Therefore the add on products of the super classes of the resource are also collected in the same way. The resource list constructed is expanded for their subclasses. As an example, if the add on product is printer, its subclasses such as "dot matrix printer" and "laser printer" should be added to the resource list.

By going back to the product descriptions, the product names corresponding to these resource types are obtained and the retailer's catalog is accessed again to extract related entries to be added to the customer's resulting catalog. In this way, a dynamic customer catalog is created for this anchor product together with its add-on products.

Three standard queries shown in Figures 13, 14 and 15 are executed against the RDF description of this catalog resource to obtain the catalog entries for the add on products:

The first of part the query given in Figure 13.b matches each catalog entry pointer "cei" in the temporary result produced by the query given in Figure 12. The second part of the query accesses the designated catalog entry and obtains the product description identifier "pdi" and its url. The third part of the query accesses the product description element and retrieves "pddetail_url" and "pddi". The fourth part of the query gathers the resource type of this product into "rrt" from the product description element pointed by "pddetail_url" and "pddi". The function call in the CONSTRUCT part of the query extracts the super classes of each selected product's resource type from the RDF descriptions. Actually this function is executed recursively to any depth since the RDF schema does not impose any restrictions on the nesting level of the "subClassOf" property. Therefore, the query is executed until no other super classes are found, that is, until the given top resource type "Product" is reached.

The following resource types are obtained as the result of the above query according to the RDF schema description given in Figure 11:
desktop, computer.

The second standard query given in Figure 14.b is executed next against these resource types. The query accesses the RDF descriptions for each product resource type to obtain related add on product types. The function call in the CONSTRUCT part, traces down the class hierarchy of the resources to obtain all the sub classes of the extracted add-on product types. In fact this function is executed recursively to any depth, similar to finding the closure set of super classes, since the RDF schema does not impose any restrictions on the nesting level of the "subClassOf" property. The following resource types are obtained as the result of the above query according to the RDF schema description given in Figure 11:
ups, printer, scanner, laser.printer, dot.matrix.printer.

The third query given in Figure 15 finds out the catalog entries for the products corresponding to these add on product types. The query accesses the product description entries and selects the products that belong to these add on product classes, and then extracts the catalog entries to construct a catalog entry pointer list for the selected add on products. Catalog entry pointers for the following items are obtained as the result of the third query:
R1-D1-M2-UPS500, R1-D1-M2-Laser6, R1-D1-M2-PR9.

As demonstrated through these examples, a customized catalog can be dynamically created by executing the given standardized XML-QL queries. Such a dynamic

```
FUNCTION Get.RDF.SuperClass(in $inpClass, in $topClass, in $res.tag.name)
   WHERE <rdf:RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
                  xmlns:rds="http://www.w3.org/TR/WD-rdf-schema#">
           <rdf:description ID = "$inpClass" >
           <rds:subClassOf  resource = "$superClass">
         </></></> IN "http://www.srdc.metu.edu.tr/sc/common.schema.rdf",
         EXPR "($superClass != $topClass)"
   CONSTRUCT <$res.tag.name>$superClass</>
   { Get.RDF.SuperClass( $superClass, $topClass, $res.tag.name ) }
END
                                  (a)

WHERE <catalog.entry.pointer ident = "$cei">
       <url.reference url.string = "$catentry_url">
      </></> IN "http://www.srdc.metu.edu.tr/sc/result1.xml",

      <catalog.entry ident = "$cei">
       <product.description.group>
        <product.description.pointer ident = "$pdi">
         <url.reference url.string = "$proddesc_url">
      </></></></> IN "$catentry_url",

      <product.description ident = "$pdi">
       <rdf.class><$rrt></></>
      </> IN "$proddesc_url"

CONSTRUCT <rdf.resource.type> $rrt</>
          { Get.RDF.SuperClass($rrt, "Product", "rdf.resource.type") }
          INTO "result2.xml"
                                  (b)
```

Figure 13. A standardized XML query to find out the resource type and the super classes of a product

catalog can be presented to the customer according to his preferred style by using the XML's style sheet mechanism.

5.3. Full automation of the supply chain

Assume that the user selects the items and puts the desktop, *R1-PC333-B*, and a laser printer *R1-D1-M2-Laser6* into his shopping cart and finalizes his shopping. The catalog agent accepts the shopping cart as part of a "purchase.msg" message, and according to the customer's preferred purchase protocol (like OBI or OTP) [1] starts a workflow process and provides an order tracking identifier so that the customer can monitor its order.

Assuming that the customer prefers OTP, an example workflow process definition invoked by the catalog agent is available at "http://www.srdc.metu.edu.tr/sc/workflows/wf.processOTP.xml". This process, through a user task, first registers the

```
FUNCTION Get.RDF.SubClass (in $inpClass, in $res.tag.name )
   WHERE <rdf:RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
                  xmlns:rds="http://www.w3.org/TR/WD-rdf-schema#">
          <rdf:description ID = "$subClass">
           <rds:subClassOf resource = "$inpClass">
         </></></> IN "http://www.srdc.metu.edu.tr/sc/common.schema.rdf"
   CONSTRUCT <$res.tag.name>$subClass</>
             { Get.RDF.SubClass($subClass, $res.tag.name) }
END
                              (a)

WHERE <rdf.resource.type> $rrt </>
         IN "http://www.srdc.metu.edu.tr/sc/result2.xml",

      <rdf:RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
               xmlns:rds="http://www.w3.org/TR/WD-rdf-schema#"
               xmlns:our="http://www.srdc.metu.edu.tr/sc/common.schema.rdf#">
       <rdf:description ID="$rrt">
        <MESChain:Added_Value ID="$addOnClass">
      </></></> IN "http://www.srdc.metu.edu.tr/sc/common.schema.rdf"

CONSTRUCT <rdf.addon.class>$addOnClass</>
          { Get.RDF.SubClass($addOnClass, "rdf.addon.class") }
          INTO "result3.xml"
                             (b)
```

Figure 14. A standardized XML query to find out the Add-On classes and their sub-class of a given resource type

customer into a customer database if s/he is not already registered, and obtains a unique customer id for the customer. The process continues checking the stock availability for each item in the shopping cart. For all the products for which in stock quantity is not sufficient, an automatic order is generated and sent to the catalog agent of the related distributor. Note that if the customer order can be fulfilled with the available stock, the process continues by requesting the payment from the customer conforming to the payment exchange procedure in OTP. After the receipt of the payment, the process confirms the delivery with the customer, and realizes the delivery of goods according to the delivery exchange procedures in OTP.

In OTP, the roles like "payment handler", "delivery handler" could be either realized by different organizations or by a single organization. The sample workflow process definition given assumes that all the roles are realized by the merchant organization.

Note that a retailer is a customer to the distributor and the same workflow process can be used by the distributor with the possible exception that the shipping details and the payment may be predetermined. Thus when the automatically generated orders of retailer's workflow process are sent to the distributors catalog agent, it in

```
WHERE <rdf.addon.class>$addOnClass</>
        IN "http://www.srdc.metu.edu.tr/sc/result3.xml",

      <catalog>
       <catalog.entry.pointer ident = "$cei">
        <url.reference url.string = "$catentry_url">
       </></> ELEMENT_AS $cep_element
      </> IN "http://www.srdc.metu.edu.tr/sc/R1.catalog.xml",

      <catalog.entry ident = "$cei">
       <product.description.group>
        <product.description.pointer ident = "$pdi">
         <url.reference url.string = "$proddesc_url">
      </></></></> IN $catentry_url,

      <product.description ident = "$pdi">
       <rdf.class><$addOnClass></></>
      </> IN $proddesc_url

CONSTRUCT $cep_element INTO "result4.xml"
```

Figure 15. A standardized XML query to find out the catalog entries of the Add-On classes of a resource type

return starts its own workflow process. In this way the automation of the whole supply chain is achieved.

5.4. Monitoring

Monitoring on the supply chain includes monitoring of status of customer orders, supply purchase and/or production orders. Monitoring requirements can be considered in the following categories:

- *Customer Order Monitoring.* A customer may monitor only her own orders one at a time by contacting the catalog agent for each order. When a customer places an order through the catalog agent, she is provided with a tracking number. Later, the customer may inquire the status of her order by contacting the catalog agent and presenting the tracking number of the order. The catalog agent sends the workflow domain manager a "wf.query" message which includes the order tracking number. The workflow domain manager identifies the related workflow process instance by using the order tracking number, and sends the status of the process back to the catalog agent. The catalog agent then forwards the status of the order to the customer.

- *Monitoring Orders at a Participant.* A supply chain participant may want to monitor a single order or all orders, i.e., the status of customer orders down the

```
WHERE <catalog>
  <catalog.entry.pointer ident = "$cei">
   <url.reference url.string = "$catentry_url">
  </></></> IN "http://www.srdc.metu.edu.tr/sc/M1.catalog.xml",
  <catalog.entry ident = "$cei">
   <product.description.group>
    <product.description.pointer ident = "$pdi">
     <url.reference url.string = "$proddesc_url">
  </></></></> IN $catentry_url,
  <product.description.general>
   <feature.set> <feature.group>
     <feature.name>Access Speed</>
     <feature.name.value><rate>$speed</></>
  </></></> IN Get.Proddesc.General.Element($pdi, $proddesc_url),
  EXPR "($speed>=32)"
CONSTRUCT <catalog.entry.pointers>
       Get.UpLink.Items("http://www.srdc.metu.edu.tr/sc/M1.catalog.xml",$cei)
       </> INTO "result5.xml"
```

Figure 16. A standardized XML query for traversing up_links

supply chain. This can be achieved by authorized users via direct connection to the participant's own workflow domain manager which maintains a run-time list of active processes and their status.

- *Monitoring the Entire Chain.* Monitoring of the entire chain can be achieved by directly connecting to the workflow domain managers of each participant on the chain. When the results are combined, it will include all the active orders and their status on the chain.

5.5. Bi-directional traversal on the supply chain

Assume a business-to-consumer scenario where a customer is willing to buy a product with a certain brand name. This can be a whole product or a subcomponent in a product. The customer prefers to refer to the manufacturer's catalog to see the available alternatives. Since s/he can only purchase from a retailer it is necessary to traverse the supply chain starting from this manufacturer until the retailers that sell the product (or the products which contain this product as a component) are reached.

As an example, consider the supply chain given in Figure 10 where a customer wants to buy a 32X or faster CD produced by manufacturer M1. We use "up_link" feature of the supply chain which is defined as an RDF property as explained in Section 4.4.

The standardized query given Figure 16 first retrieves the 32X or faster CDs produced by manufacturer M1 and then traverses the up links to find out the retailers

```
FUNCTION Get.UpLink.Items (in $catalog_url, in $cei )
/* When a retailer catalog has been reached, the following query will
   terminate and return a catalog entry pointer to a retailer product*/
{  WHERE <catalog RDF="$rdf_url"></> IN $catalog_url,
     <rdf:RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
              xmlns:dc="http://purl.org/metadata/dublin_core#">
      <rdf:Description about="$catalog_url">
      <dc:Subject>"Retailer"</>
     </></> IN $rdf_url,
     <catalog>
       <catalog.entry.pointer ident = "$cei">
       </> ELEMENT_AS $cep_element
     </> IN $catalog_url
  RETURN $cep_element }
/* This it is not a retalier catalog,
   follow the "uplink"s until a retailer catalog is found.*/
{  WHERE <catalog RDF="$rdf_url"></> IN $catalog_url,
     <rdf:RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
          xmlns:msc="http://www.srdc.metu.edu.tr/sc/common.schema.rdf#">
      <rdf:Description about="$catalog_url">
      <msc:up_link resource="$up_catalog_url"></>
     </></> IN $rdf_url,
     <catalog>
      <catalog.entry.pointer ident = "$up_cei">
       <url.reference url.string = "$up_catentry_url">
     </></></> IN $up_catalog_url,
     <catalog.entry ident = "$up_cei">
      <product.description.group>
       <product.description.pointer ident = "$up_pdi">
        <url.reference url.string = "$up_proddesc_url">
     </></></></> IN $up_catentry_url,
     <product.description ident="$up_pdi">
      <product.description.general.pointer ident="$cei">
     </></> IN $up_proddesc_url
   RETURN Get.UpLink.Items( $up_catalog_url, $up_cei ) }
END
```

Figure 17. The recursive XML-QL function used in the query in Figure 16

selling this product. Assume the following item is found in M1's catalog: *M1-CD32*

The function call in the CONSTRUCT part, traces the up links to find out the retailers selling this product. The function definition in Figure 17 obtains the URL of the RDF descriptions from <?xml ...> as proposed in Section 3.4. Once the RDF description is obtained, the query checks the SUBJECT element to see if a retailer catalog is reached. Note that the SUBJECT element of Dublin Core contains the roles of participants on the supply chain as proposed in Section 4.1. If a retailer's catalog is not reached yet, the second query block of the function executes to obtain the up links of the current catalog and searches the up link

```
Get.Uplink.Items(M1.catalog,"M1-CD32")       Get.Uplink.Items(D1.catalog,"D1-M1-CD32")
uplinks: D1,D2                               uplinks: R1,R2
Construct:  D1.catalog, "D1-M1-CD32"         Construct:  R1.catalog, "R1-D1-M1-CD32"
            D2.catalog, "D2-M1-CD32"
               (a)                                              (b)

Get.Uplink.Items(R1.catalog,"R1-D1-M1-CD32") Get.Uplink.Items(D2.catalog,"D2-M1-CD32")
uplinks: R1,R2                               uplinks: R1,R2
returns: R1.catalog, "R1-D1-M1-CD32"         Construct:  nil
               (c)                                              (d)
```

Figure 18. Trace of the standardized XML query given in Figure 16

catalogs to find the item (or the products containing the item) recursively. This function requires both a CONSTRUCT and a RETURN clause since there could both be several up links to be traversed and also several products containing the required item. If only a RETURN statement is used the function will return upon a first match. To be able to get the complete combinations, the CONSTRUCT statement is used together with a RETURN clause where the RETURN clause executes only after the CONSTRUCT clause produces all combinations.

The execution of the query continues with the recursive calls of the function as follows: The function is called for "M1.catalog, M1-CD32" as shown in Figure 18.a which constructs "D1.catalog, D1-M1-CD32" and "D2.catalog, D2-M1-CD32" by following the up links D1 and D2. The function in the construct clause is called for "D1.catalog, D1-M1-CD32" as in Figure 18.b and it constructs "R1.catalog, R1-D1-M1-CD32". We assume that there are no matching items in R2's catalog. The function in the construct clause is called for "R1.catalog, R1-D1-M1-CD32" as in Figure 18.c and which returns a catalog entry pointer since it finds out that R1 is a retailer. The call in (b) terminates by the termination of (c), which causes the construct part of (a) to call the function once more for "D2.catalog, D2-M1-CD32" as in Figure 18.d. The function finds the up links R1 and R2, but we assume that these catalogs do not have any matching items. Therefore the functions returns nil, which causes the call in (a) to terminate and return the final result as *"R1.catalog, R1-D1-M1-CD32"*.

5.6. On-the-fly supply chain construction

On-the-fly supply chain integration is greatly facilitated with the proposed architecture. Since the meta data of the electronic catalogs are described through RDF with Dublin Core, the discovery of them by search agents becomes possible. In a business to business scenario, once a buyer search agent locates an electronic catalog of a potential supplier, it informs its own catalog agent which in turn contacts and negotiates with the catalog agent located. If a deal is reached, each catalog agent starts an approval process to evaluate the new candidate partner for its own company. If both parties approve each other as 'new partners', then the catalog agents exchange a "new.partner.msg" message to automatically insert "up_link" and "down_link" property values pointing to each other in the RDF descriptions of the catalogs. They also exchange the necessary CBL data such as market participant information as part of the "new.partner.msg" operation.

The assumption over here is that the electronic catalogs conform to CBL. For those catalogs that do not conform to CBL or that are kept in legacy databases, how to map them to CBL catalogs is discussed in Section 4.2.

6. Contributions of the work

The contributions of the work described in this paper are as follows:

1. *Catalog interoperability.* MESChain proposes XML CBL catalog DTDs as the canonical data model for integration of heterogeneous catalogs. By using the EXECUTE instruction introduced in Section 3.1, it is possible to integrate XML fragments obtained from external applications through wrapper programs. In this way, XML documents dynamically generated from a number of possibly heterogeneous resources can be flexibly integrated. This integration is done at the document level rather than at the code level. With this approach, an organization may continue to keep its product specifications and catalog information in its legacy applications and still benefit from the supply chain architecture proposed.

   When all the catalogs are available in XML conforming to standard DTDs, catalog integration and customized catalog generation reduce to issuing standardized XML queries as demonstrated in Section 5.1.

2. *Given an anchor product on the supply chain, discovery of the items and services that add value to this product.* The product taxonomy and an "Added_Value" property are defined in RDF Schema Definition Language, more specifically through the "subClassOf" and "property" features of the RDF.

   The "subClassOf" mechanism makes possible to give a product taxonomy, and through the "property" feature of RDF, it is possible to state which resource (e.g. a printer) is an add on product for which other resource (e.g. a computer). This mechanism is explained through examples in Sections 4.3 and 5.2.

3. *Bi-directional traversal on the supply chain.* The relationships among the catalogs on the supply chain, in other words participant roles, are specified through the "up_link" and "down_link" descriptions in RDF which facilitate the bi-directional catalog search and integration. This mechanism is explained through examples in Sections 4.4 and 5.5. Catalog agents help to establish the "up_link" and "down_link" specifications for on-the-fly supply chain construction as explained in Sections 4.5 and 5.6.

4. *An open architecture capable of supporting different electronic commerce standards.* It is widely believed that a single dominant ecommerce standard is unlikely. Rather there will be many standards. An ecommerce architecture should be open in the sense that it should be able to support more than one standard at a time. As an example both OBI and OTP are different but similar standards. When the base is XML, since XML is machine processable, it naturally follows that the task of supporting more than one standard should be delegated to the agents and this is the approach taken in MESChain by associating catalog agents to catalogs.

5. *Resource discovery on the Web.* Resource discovery on the Web has been an important yet difficult problem. It is considered in two categories in our architecture: discovery of the electronic catalogs and discovery of the types of products contained in a catalog.

   For the discovery of electronic catalogs, the meta data of a catalog is expressed in RDF using the Dublin Core element set. For the discovery of product types, a catalog schema conforming to CBL and a product taxonomy expressed in RDF are used. Having a standardized catalog schema and product taxonomy, it becomes possible for resource discovery agents to find out these catalogs and types of products in them as explained in Section 4.1.

6. *Automating the whole process on the supply chain.* When a catalog agent receives a customer order, it identifies the associated protocol and initiates the corresponding workflow template instance. The companies on the chain can define their workflow templates in XML conforming to "workflow.dtd" proposed and a workflow engine in Java is provided to execute these definitions.

   We believe the workflow architecture proposed in Section 4.6 better fits to supply chain architecture for the following reasons:

   - The use of XML for process definitions. In this way, process definitions become highly transportable and interoperable. Any organization may develop its own processes, in addition to sharing common business processes developed by others without any difficulty. This brings re-usability to the process definitions and provides high productivity.
   - Processes can run platform-freely, since the engine is coded in Java. The engine is capable of executing tasks locally or remotely.
   - The use of XML messages to communicate with the workflow domain manager. This enables remote control of the workflow manager, for example initiating a process from the library, or inserting a new process definition into the library. With proper coordination of authorization requirements, it is also possible to initiate processes across organization boundaries. Since initiating a process remotely involves only sending the "process activation" message, this message can be sent to the workflow domain manager of any organization (in our architecture), to actually initiate the corresponding

process. At the termination of this process, the originator will be informed by a response message in XML. That is, the proposed workflow architecture is capable of running complex processes that involve more than one organization in a supply chain.

- The adaptive architecture of the workflow system supports dynamic changes on the running process instances in case of external changes in the network or unexpected conditions in the organization. In such a situation, running process instances may selectively be modified as needed. The modification may cause rolling back some completed tasks which will be compensated as part of the modification, and execution of the modified instance will proceed from thereon.

7. Related work

The most notable work on supply chain automation and integration come from two main industry consortiums, CommerceNet consortium and the RosettaNet consortium. The CommerceNet is the leading industry association for electronic commerce and in [9] describes the required properties of electronic catalogs for supply chain integration. This document motivated the work presented in this paper. RosettaNet project [34] on the other hand, stresses the importance of open content and open transaction standards for supply chain integration and is producing the standard descriptions for computer industry as XML DTDs.

A more recent electronic commerce interoperability infrastructure proposed by CommerceNet is the eCo framework [10]. In the eCo framework businesses agree on a common method of describing what they do, rather than the standards of what they do and how they do it. The eCo framework consists of an architectural specification and a semantic specification. The eCo Architectural Specification presents information about an e-commerce system in seven different categories (layers). The eCo Semantic Specification provides a sample set of business documents that can be used inside the eCo framework. These can be used as is, or extended and modified to meet specific needs [10]. In the network layer the eCo compliant markets are listed like computers, phones, or books. In the market layer, for a specific market like computers, their participating businesses are listed like Dell, or IBM. In the business layer, the services provided by a business are listed for example purchase or rent. At the service layer, the possible interactions are listed where each interaction defines input, output documents and optionally an execution URI. As an example the interaction for purchasing a computer can be an order document as input, and an invoice as output defined in XML using the corresponding CBL DTDs [11]. The document layer specifies the structure of a document as well as listing data elements if there are any. At the data element layer, details of data elements are presented.

WISE system [2] describes an infrastructure for business to business electronic commerce. This infrastructure includes an "internet workflow engine" acting as the underlying distributed operating system controlling the execution of business processes, a process modeling tool for defining and monitoring the processes, a catalog tool for virtual enterprise services. In WISE, virtual business processes are constructed by using the services offered by different companies as building blocks. The WWW catalog provided uses Java Applet/Servlet technology to allow companies in the trading community to advertise and to see the semantics of the services provided by other companies.

CrossFlow [12] aims at providing high-level support for workflows in dynamically formed virtual organizations. Virtual organizations are formed when some activities of a company is outsourced to external service providers. High level support is obtained by abstracting services and offering advanced cooperation support. Virtual organizations are dynamically formed by contract-based match-making between service providers and consumers.

MESChain differs from WISE and CrossFlow as follows: MESChain addresses the supply chain automation and catalog interoperability by exploiting the recent Web standards and using workflow and agent technology, rather than constructing a workflow from the building blocks provided by companies in the trading community.

Related with electronic catalogs, [13] discusses issues and solutions about the creation of a product information database as a foundation for deploying an electronic catalog. In the proposed architecture, an HTML form will be dynamically generated from the database when a buyer wants to browse the catalog through the Internet. To exchange product information among trading partners, it is suggested that the contents of product information database be either electronically sent in XML or this information be queried online. For example when a distributor needs the product information from the manufacturer, the manufacturer should either send this information in XML format or should let its product database to be queried online by the distributor. The latter relives the distributor from maintaining product information which is primarily maintained in the manufacturer's database.

In MESChain, a participant has the option of keeping its own product information database and generating the electronic catalog in XML from this database dynamically by the EXECUTE instruction. Sharing the product information among participants is also possible in MESChain by using the "product.description.pointer" element of the CBL catalog schema which provides for online querying to retrieve product information from another participant.

Anderson Consulting's BargainFinder [3] is one of the well-known and early example of electronic marketplaces. BargainFinder was the first shopping agent for on-line price comparison. It requests its price from each of nine different merchant web sites as if it is using a web browser to do the request. Although it was a limited prototype, it offered many insights into the issues involved in price comparisons.

OFFER [4] is an electronic brokering architecture which uses OMG's CORBA as a distribution infrastructure. There are three main components: suppliers, customers and e-brokers. A customer can search for a service either directly in the e-catalog of the supplier or use the e-broker to search all the e-catalogs of all the suppliers, which are registered with this broker. CORBA is chosen as the communication infrastructure to solve the interoperability problem. They specify a standard IDL interface for the e-catalogs of a supplier and for the e-broker. Each supplier is responsible for implementing this interface to be able to register with the broker and for others to find its catalog. As the negotiation mechanism e-brokers employ simple auction mechanisms.

Related with describing resources for information discovery, the Information Manifold (IM) [24] project developed at AT&T Bell Labs., provides a uniform interface to the resources. The approach requires the participating sources manually describe their available content and querying capabilities using the content and capability records defined in the IM project. When a complex query arrives, it is divided into sub queries such that each sub query can be send to an appropriate source, and answers from these sources are combined to form the resulting answer of the complex query. The content and capability records developed does not conform to any standard.

In MESChain, the contents of a catalog is expressed in XML conforming to CBL and product taxonomy in RDF, both of which can easily be queried via XML-QL or XSLT.

There are many workflow management systems, some of which are briefly described in the following:

METUFlow [15] is a fully distributed workflow management system that uses a block structured process specification language called MFDL. A process tree is generated from a process definition, and it is used for guard generation for each task in the process definition. The guards are used by task handlers to control the activation of the user tasks at run time. Distributed execution of METUFlow is based on CORBA. METUFlow's highly distributed architecture makes dynamic modification very inefficient. In MESChain, however, the workflow architecture by keeping all the process information in a workflow process object, efficiently handles the dynamic modifications.

Another distributed workflow management system is METEOR (Managing End to End Organization) [36]. Its workflow execution model is driven by inter-task dependency rules that are expressed in a specifically designed script language. METEOR allows workflow definition at two levels of abstraction using two different languages: the Workflow Specification Language, and the Task Specification Language. Process definitions are saved in an intermediate format that is used for automatic code generation at run time. The runtime code generators output code for task managers and task invocation, data object access routines and recovery mechanism. The generated code includes all the inter-task dependencies required by the definition of the process, and it is based on CORBA and Web environment for distributed execution.

In [28], the use of Web technology for workflow is presented with METEOR2 Web-based workflow management system (WebWork). WebWork is said to be web-based rather than web-enabled since both interfaces and communication/ distribution infrastructures are built using Web technology. Data flow is realized through exchanging HTML pages and CGI is the main communication mechanism with servers.

There is also some previous work on realizing a workflow system with the use of agents. DartFlow workflow management system [5] uses Web-browser embedded Java applets as its front end and transportable agents as the backbone. A transportable agent is a program that migrates machine to machine in a heterogeneous network. In DartFlow, each business process can be handled by an agent. Agent Tcl system is used to implement transportable agents. Since agents in DartFlow do not use a standard communication language, its usage is limited to those who make use of Agent Tcl system.

8. Conclusions

Electronic commerce is one of the most exciting and fast moving fields of today with a high demand for innovative new technologies and interoperability standards [1, 16, 17]. The progress and wider dissemination of electronic commerce will be possible with interoperable architectures which allow consumers and businesses seamlessly and dynamically come together and do business without ad hoc and propriety integration. In this paper we describe such a supply chain integration architecture that provides for the expected functionality by exploiting and integrating the current interoperability standards for the Web.

The experiences we have gained through the implementation of MESChain indicated that current enabling technologies lack certain features necessary for supply chain integration. We propose some extensions to meet the required functionality and also demonstrate how to fit these technologies together. For example, the EXECUTE instruction introduced to XML provided very helpful in dealing with catalogs maintained in legacy databases or applications.

In the architecture we propose, the canonical data model for catalog integration is the CBL catalog DTDs. In future more than one catalog schema may be supported to give businesses a choice. As long as the supported schemas are limited, automatic conversions among them can be possible.

Acknowledgements

The authors wish to gratefully acknowledge the METU SRDC team for implementing the parts of the system: *Yusuf Tambag* for the catalog agent, *Necip Oner Hamali* for the extended XML-QL query processor, *Yalin Yarimagan* for the workflow system, and *Ender Sevinc* for the EXECUTE instruction.

Notes

1. We prefer to call protocols like OBI and OTP "purchase protocols" rather than "electronic commerce" protocols since electronic commerce involves other activities like finding the related catalogs and comparing the products.

References

1. N. Adam and Y. Yesha, "Electronic Commerce: Current Research Issues and Applications", Springer, 1996.
2. G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt and N. Weiler, "WISE: Business to Business E-Commerce", hhtp:// www.inf.ethz.ch/department/IS/iks/research/wise.html.
3. Bargain Finder URL. http://bf.cstar.ac.com/bf.
4. M. Bichler, C. Beam, and A. Segev. OFFER: A broker-centered object framework for electronic requisitioning. In IFIP Conference "Trends in Electronic Commerce '98.
5. T. Cai, P. A. Gloor, and S. Nog. "DartFlow: A workflow management system on the web using transportable agents." Technical report, Dartmouth College, 1997.
6. I. Cingil, A. Dogac, E. Sevinc and A. Cosar "Dynamic Modification of XML Documents: External Application Invocation from XML" ACM SIGecom Exchanges, Vol. 1, No. 1, August 2000, pp.1-6.
7. I. Cingil, "An Architecture for Supply Chain Integration and Automation on the Internet" PhD Thesis, Middle East Technical University, in preparation.
8. Commerce XML Resources, http://www.cxml.org/home.
9. CommerceNet, Catalogs for the Digital Marketplace, Note 97-03, 1997.
10. CommerceNet, "eCo architecture for for electronic commerce interoperability" The eco Framework, http://eco.commerce.net
11. CommerceOne Inc, http://www.commerceone.com/ XML/CBL, 1997.
12. CrossFlow, "Cross-Organizational Workflow Support in Virtual Enterprises", ESPRIT Project 28635, http://www.crossflow.org.
13. S. Danish, "Building Database-driven Electronic Catalogs", ACM Sigmod Record Special Section on Electronic Commerce. 27(4), December 1998.
14. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, "XML-QL: A query language for XML", W3C Document, http://www.w3.org/ TR/NOTE-xml-ql, 1998.
15. A. Dogac et al, "Design and Implementation of a Distributed Workflow Management System: METUFlow", Springer Verlog, 1998.
16. A. Dogac, Guest Editor, ACM Sigmod Record Special Section on Electronic Commerce. 27(4), December 1998.
17. A. Dogac. Guest Editor, Distributed and Parallel Databases, Special Issue on Electronic Commerce. April 1999, Vol.7, No.2.
18. DOM, "Document Object Model Level 1 Specification", W3C Recommendation, http://www.w3.org/ TR/REC-DOM-Level-1, 1998.
19. Dublin Core, "Dublin Core Metadata Element Set", http://purl.org/DC/, 1998.
20. R. Glushko, J. M. Tenenbaum and B. Meltzer, "An XML Framework for Agent-based E-commerce", Communications of the ACM, 42(3), 1999.
21. JavaScript, "JavaScript Documentation", http:// developer.netscape.com/docs/manuals, 1998.
22. Y. Labrou and T. Finin, "A proposal for a new KQML specification", Technical Report TR-CS-97-03, University of Maryland, 1997.
23. S. St. Laurent, "XML: A Primer", MIS Press, 1998.
24. A. Levy, A. Rajaraman and J. Ordille, "Querying Heteregeneous Information Sources Using Source Descriptions", Proc. Int. Conf. on Very Large Data Bases, India, 3-6 September 1996.
25. LotusXSL, "Implementation of the XSL Transformations (XSLT) and the XML Path Language (XPath)", http:// www.alphaworks.ibm.com/tech/LotusXSL, 1999.
26. B. Meltzer and R. Glushko, "XML and electronic commerce: Enabling the network economy", In [16].
27. E. Miller, " An Introduction to the Resource Description framework", D-Lib Magazine, 1998.
28. J. Miller, D. Palaniswami, A. Sheth, K. Kochut, and H. Singh. WebWork: METEOR2's web-based workflow management system. Journal of Intelligent Information Systems, 10(2):1–30, 1998.
29. OBI, "Open Buying on the Internet", http:// www.openbuy.org/, 1998.

30. Oracle, "XML Support in Oracle8i", http:// www.oracle.com/xml/documents/xml_twp/, 1999.
31. OTP, "Open Trading Protocol", http:// www.otp.org/, 1997.
32. RDF Schema, "Resource Description Framework (RDF) Schema Specification", W3C Proposed Recommendation, http://www.w3.org/TR/PR-rdf-schema, 1999.
33. RDF Syntax, "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, http://www.w3.org/TR/ REC-rdf-syntax, 1999.
34. RosettaNet, http://www.rosettanet.org/general/ finished-project/laptop.html, 1998.
35. E. Sevinc, "Dynamic Modification of XML Documents" MS Thesis, Middle East Technical University, 2000.
36. A. Sheth and K. J. Kochut, "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems" NATO, ASI, 1997.
37. D. Suciu, "Semistructured data and XML", Proc. Int. Conf. on Foundations of Data Organization, 1998.
38. M. Woolridge and N.R. Jennings, "Intelligent agents: theory and practise", The Knowledge Engineering Review, 10(2), 115-152, 1995.
39. Workflow Management Coalition, Glossary: A Workflow Management Coalition Specification., WfMC Standart, WfMC-TC-1011, 1994.
40. XML, "Extensible Markup Language (XML) 1.0", W3C Recommendation, http:// www.w3.org/ TR/REC-xml-19980210, 1998.
41. XML4J, "XML Parser for Java", http:// www.alphaworks.ibm.com/tech/xml4j, 1998.
42. XSL, "Extensible Stylesheet Language (XSL) Version 1.0", W3C Working Draft, http:// www.w3.org/ TR/xsl, 2000.
43. XSLT, "XSL Transformations (XSLT) Version 1.0", W3C Recommendation, http:// www.w3.org/ TR/xslt, 1999.
44. Y. Yarimagan, "A Component Based Workflow System for Enacting Process Defined in XML", MS Thesis, Middle East Technical University, Dec 1999.