End-to-end E-commerce Application Development Based on XML Tools.

ARTICLE · JANUARY 2000

Source: DBLP

10 AUTHORS, INCLUDING:

Kostas Kontogiannis
National Technical University of Athens
174 PUBLICATIONS **2,570** CITATIONS
SEE PROFILE

John Mylopoulos
University of Toronto
667 PUBLICATIONS **18,177** CITATIONS
SEE PROFILE

End-to-end E-commerce Application Development Based on XML Tools

Weidong Kou
David Lauzon
William O'Farrell
IBM Toronto Lab, Canada

Teo Loo See
Daniel Wee
Daniel Tan
Nanyang Polytechnic, Singapore

Kelvin Cheung
Richard Gregory
Kostas Kontogiannis
University of Waterloo, Canada

John Mylopoulos
University of Toronto, Canada

Abstract

In this paper, we discuss an electronic business application framework and its related architecture. The framework is presented in the form of a prototype system which illustrates how XML tools can assist organizations on building and deploying e-commerce applications. The use of the system is presented in the form of a sample e-commerce application that involves shopping in a virtual store. The framework allows for customers and service providers to be linked through XML/EDI with back-end applications and to trigger e-procurement orders which are sent using either XML or EDI messages. Invoice handling and order tracking are also discussed along with extensions to the proposed architecture that allow for business process logic to be encoded at the server side, in the form of Event Condition Action scripts.

1 Introduction

E-business has become the focal point in the transformation of key business processes through the use of Internet technologies. By incorporating Internet technologies into core business processes, organizations are able to integrate back-end software applications and, data base systems into a seamlessly open environment [7], [6]. At the core of e-business lies electronic commerce or e-commerce, which is defined as commerce activities conducted electronically, particularly over the Internet, from online transactions to streamlined billing and payment systems.

Recent reports from investment and consulting firms (such as Goldman Sachs and First Data International) indicate that business-to-business (B2B) e-commerce related revenue over the next five years, can be as high as one trillion US dollars. E-commerce is with no doubt driving the fast growing digital economy in the new millennium.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

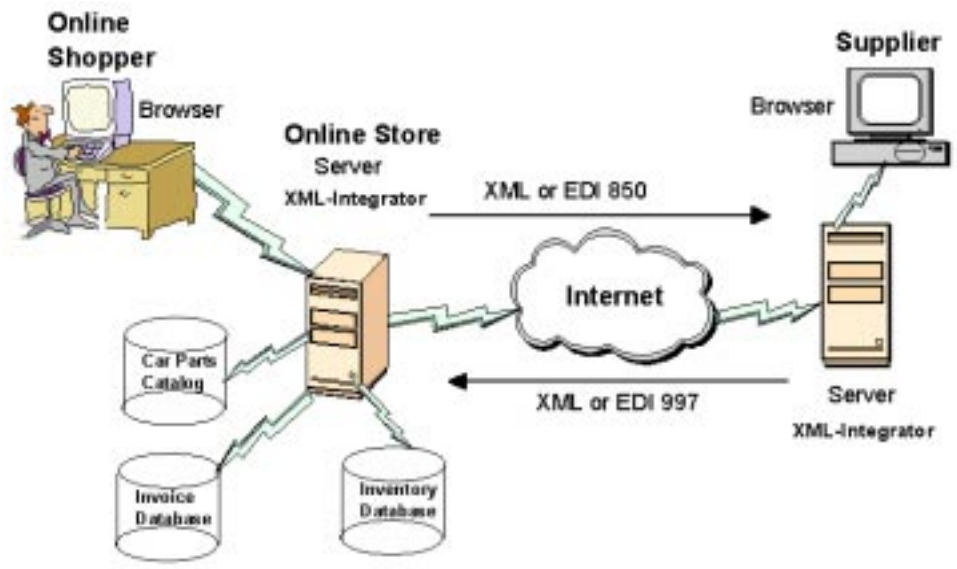

Figure 1: An e-Commerce Application Demo.

To meet the rapid growth of e-commerce, new technologies have emerged and new standards have been set. One such standard is XML (extensible markup language) [5] that provides an excellent vehicle to move data between applications and to allow for a smooth integration of new e-commerce applications from small and medium enterprises (SME), without requiring too much investment on complex or expensive EDI-based infrastructure.

Key e-commerce application areas such as inventory analysis, sales, accounting, and purchasing are only part of a larger framework that supports the selection, purchase, and delivery of a product to a customer. This framework can be modeled as a collection of information flows from one point to the next in the business process chain.

In this report, we present a prototype infrastructure that allows for the development of end-to-end e-commerce applications [2] using XML tools, and IBM's Websphere server environment. The motivation is to design and develop an architecture that allows different business partners to work together in a flexible way and, to deploy end-to-end e-commerce applications both in the business-to-business (B2B) and business-to-consumer (B2C) arena.

2 Prototype System Description and Discussion

A prototype implementation of an end-to-end e-commerce application was demonstrated at CASCON-1999, an annual conference co-sponsored by IBM and the Canadian National Research Council, in November 8-11, 1999 [2]. This prototype is based on a set of Internet technologies, including IBM XML tools, and Websphere, EJBs (enterprise Java Beans), and Java Servlets. The system aims to provide infrastructure for developing e-business applications by composing distributed web-enabled components.

The proposed environment focuses on typical e-commerce applications (Figure 1), in which multiple shoppers can browse a catalogue of an online store, select products, add them to a shopping cart, and place orders. The system provides integration with the underlying inventory system and, depending on the inventory information, a purchase order can be transmitted to a supplier in either XML or EDI (Electronic Data Interchange) 850

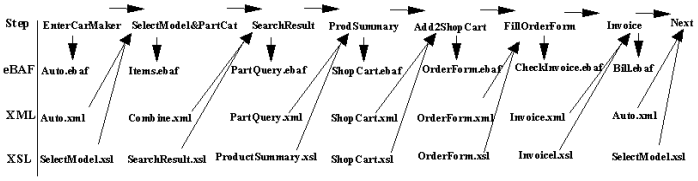

Figure 2: Flowchart of Shopping Steps And Associated XML, XSL, and eBaf Files.

Figure 3: Home Page of the Car Parts Store.

format. The supplier responces can include acknowledgment messages in either XML or EDI 997 format. The online store can also track orders, bill the shoppers, and contact a transport company for delivery.

The prototype environment was built with IBM XML tools that are currently being developed at IBM [1] as part of the Electronic Business Application Framework (e-BAF). The e-BAF framework defines files which contain SQL scripts for an XML Integrator Servlet, data parameters provided by the user and, XSL style sheets for the presentation of results to the user in HTML.

The flowchart in Figure 2 llustrates each step that a shopper will go through with its associated XML, XSL, and e-BAF files.

1. Enter the AutoPartsMart Virtual Mall

   Figure 3 llustrates the home page of the demo for the Auto Parts Mart. It consists of an HTML page. The shopper selects the model of the car for which he would like a list of parts.

   Upon the selection of the manufacturer, this information is captured in an XML fragment of the form:

```
<?xmlversion="1.0"?><Input><XMLProject><Directory>\samples\NYPAutoMart
</Directory><EBAFile>Dauto</EBAFile></XMLProject><Fragment><Manufacturer>
Honda</Manufacturer></Fragment></Input>
```

   The fragment is sent to the XML Integrator Servlet running on the WebSphere Application Server. The fragment consists of the Project directory path for this demo, the e-BAF file which contains the script for the XML Integrator, and the data fragment that contains the parameters captured from the shopper.

   The XML Integrator Servlet parses the information in the XML fragment using the XML4J parser. The LotusXSL processor executes the script specified in the e-BAF file with the relevant data from the shopper that is captured in the data fragment. Queries to the database can be captured in the script and the result set is output in XML format by the XML Integrator Servlet. The final output from the XML Integrator is an HTML page that is composed from an XSL and an XML file. Refer to Figure 4 below.

   The DAuto.EBAF file is parsed by the XML Integrator Servlet. Data such as the selected car model is stored via the <SessionStore> tag where an SQL Query is made to the PARTS database to retrieve the model for the auto maker, the accessories sold by the Auto Mart, and the year of the model. The result of the query is captured in an XML file and the XML Integrator Servlet is then instructed to produce the final HTML file from this XML using the XSL file specified in the EBAF file. A sample of the HTML file is shown in Figure 5.

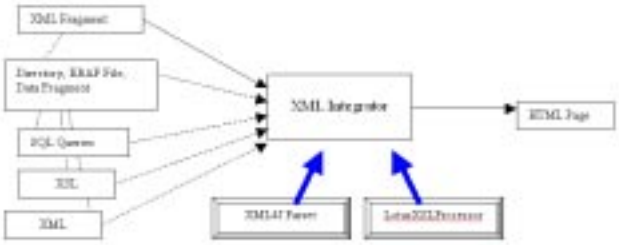

Figure 4: Input-output relationship of XML/XSL files and HTML page.

Figure 5: Logical relationship of files for ”Select Model and Car Part” step.

Figure 6: Screen of “Select Model and Car Part” Step.

Figure 7: Screen of “Product Summary” Step.

2. Select Model and Car Part

   In this step, illustrated in Figure 6, the shopper selects the model of his or her car and the part of interest, based on the car model data retrieved from the parts database. On clicking the “Submit” button, another XML fragment is sent to the XML Integrator. The flow (and subsequent flow) is similar to the previous step except that in the e-BAF file, the data fragments are now different. The processing is specified in the e-BAF file and in this case, it is Items.EBAF (see Figure 2). In this scenario, two queries have been initiated to the PARTS database in order to retrieve information as required by the Shopper. The results of the two queries are kept in two separate XML files and the XMLMerge tag is used to merge both files together. Finally, the information is presented to the shopper on an HTML page from this merged XML file using the specified XSL file.

3. Search Result

   The shopper chooses from the Search Result Page the part that is sought. The part number is captured and inserted into the fragment and sent to the XML Integrator Servlet. The PartQuery.EBAF file is used for specifying the queries to the PARTS database for retrieving the details regarding the selected parts. The results are merged together to form the Product Summary page (Figure 7).

4. Product Summary

   The shopper obtains as the result of the process the Product Summary page (Figure 7), which provides the details of the selected parts and accessories. On clicking the ”Add to Shopping Cart” button, another XML fragment is sent to the XML Integrator Servlet and the ”Shopping Cart” Page is displayed.

5. Add to Shopping Cart

   The shopper selects the number of tires that he wishes to purchase. The sub-total will be calculated for him. On

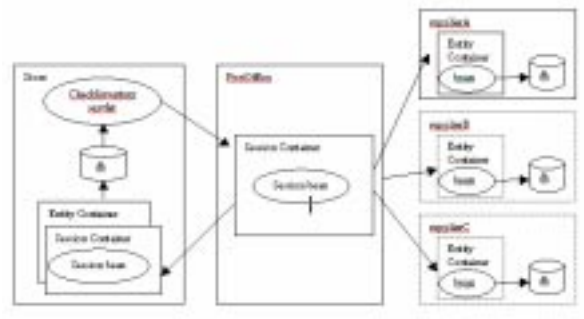

Figure 8: EJB for Information Transmission Between the Online Store and the Suppliers.

Figure 9: X12 850 Message to Supplier.

selecting Check Out, the XML fragment is sent to the XML Integrator Servlet. The Order Form is displayed to the shopper as shown in the next section with the session data pertaining to his purchase.

6. Fill in Order Form

   The Order Form shows the order details that has been provided by the shopper. The form is validated before the order can be placed. On clicking the "Place the Order", the XML fragment is sent to the XML Integrator Servlet, with the CheckInventory.EBAF in the EBAFile tag.

   In the CheckInventory.EBAF, the script uses the <Service> tag to invoke a service, CheckInventory, to perform replenishment of the PARTS database when the quantity-on-hand falls below the reorder level of the product after the purchase. For the demo, the CheckInventory service is implemented as a servlet that loads at startup and it registers itself with the XML Integrator's Service Handler. The CheckInventory Servlet invokes an Enterprise Java Bean (EJB) on the supplier server of the product when replenishment for the product is required. EJBs are used in this system to demonstrate how to simplify the process of developing the enterprise-class application systems. With EJBs, one could concentrate on writing the business logic rather than the complex middle-ware interfaces. Figure 8 illustrates its usage.

   The CheckInventory servlet reads from the database to determine the supplier address. The servlet then instantiates an EJBComms object (a simple utility class provided for this prototype) and provides it with the supplier node name and the filename that contains the X12 850 (Purchase Order) stream of data. The transmission is handled by calling methods of this class, namely, getConnection, activateBean and exitConnection. These methods establish connection with the postoffice (this can be the supplier itself) and remotely run a method of a session EJB, thereby transmitting the purchase order information. For persistence, the respective supplier Entity bean is being activated. The post office session bean will also trigger another connection with the online store to pass back the Functional Acknowledgment message X12 997.

   For the purpose of this prototype, the session bean will provide an asynchronous task to display the messages when they are received at the respective location. These are shown in Figures 9 and 10, respectively. As this is only a prototype, no business logic has been developed at the supplier end. The stream of data is not parsed at all and it is stored directly into the database.

   The invoice is prepared for the shopper and displayed as shown in Figure 11.

7. Invoice

   The shopper takes note of the Invoice Number for the purpose of order tracking at a later stage. On clicking the "Click OK' button (see Figure 11), the XML fragment is sent to the XML Integrator Servlet and the shopper is shown the HTML page to select the model and car part at the Auto Parts Mart, as a means of indicating to the shopper that he or she can make another search for another category of part if he or she wishes.

Figure 10: X12 997 Message to Buyer.

Figure 11: Screen of “Invoice” step.

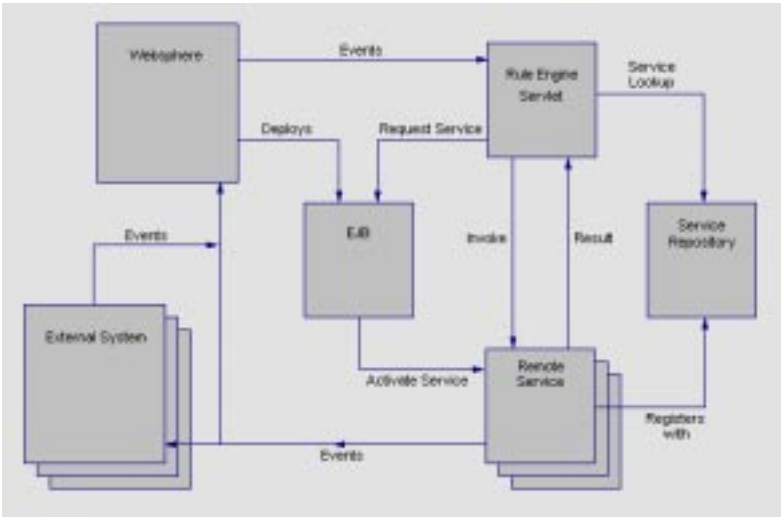

Figure 12: System Architecture.

3 Future Direction

The prototype presented in the previous section is based on pre-defined e-BAF files and on the XML Integrator. Such an architecture does not provide the kind of flexibility and extensibility which one expects in future e-commerce environments.

Figure 12 illustrates a generic architecture in which the XML Integrator presented in Figure 4 is replaced with a script language and a task enactment rule-based inference engine. Hard-coded static invocations to the various services employed in the prototype system are replaced with a script whereby business logic is encoded by a set of event-condition-action rules. The task enactment engine allows for rules to be triggered by accepting incoming events, which in turn may cause any number of rules to be triggered. For each rule triggered, a condition clause is checked and, if it is satisfied, the respective actions are invoked. In this way business process logic can be customized in the e-commerce environment to accommodate particular transaction scenaria or particular customer requirements.

In such an environment, control Integration deals with the composition, localization and invocation of remote services and components. Control Integration typically involves invoking a remote service. A service is defined

informally as a method or program, that is, something that can execute either locally or remotely and may return a value. An essential point is that services can exist anywhere on a network or the Internet. Moreover, they execute at the system on which they reside or on a thin-client and they may be implemented in any language and run on any platform. This is in contrast to other paradigms whereby code is down-loaded and executed on the client side. Events are implemented as HTTP requests that encode the details of the events in XML format. Each rule contains components that have several typed parameters along with any other information pertaining to the events upon which the rules rely. This approach allows for a Web server to be used for capturing the events across the network and processing these events by a servlet. Another advantage with this architecture is that it allows for integration with other systems by translating events into a form other environments can use.

```
<ECAScript name="B2B Example">
  <ECARule name="Parts List Query">
    <Events> <EventExpr>
        <Event name="NeedPartList">
          <Param type="ModelName" name="Model" value=""></Param>
             <!--Note that "value" will be given a value (an-->
             <!--SQL query represented in XML at runtime    -->
        </Event>
    </EventExpr> </Events>
    <Conditions> </Conditions>
    <Actions>
      <Action name="Database Query">
        <Param name="Model"></Param> </Action>
             <!--The value given above will be used here-->
    </Actions>
  </ECARule>
  <ECARule name="Part out of Stock">
    <Events> <EventExpr>
        <Event name="NotInStock">
          <Param type="PartId" name="Part"></Param> </Event>
      </EventExpr> </Events>
    <Conditions> </Conditions>
    <Actions>
      <Action name="PlaceOrder">
        <Param name="Part" value = ""></Param>
        <Param name="Supplier" value="Toronto Car Parts"></Param>
            <!--The first value will be bound at runtime,-->
            <!--the second, when the script is read      -->
      </Action> </Actions>
  </ECARule>
</ECAScript>
```

Figure 13: Example ECA Rules in XML.

The scripting language allows tasks be specified as actions in a rule-based environment that implements the Event-Condition-Action (ECA) paradigm [4]. The scripts compose remote services in a way that customizes the business transaction logic in a B2B e-commerce application. XML is used for encoding the ECA scripting language, an example of which is provided in Figure 13. The service registration system is used for determining how the actions of this script are invoked. Finally, the rule engine is based on a forward-chaining activation mechanism that binds rule premises (Events), evaluates conditions and, triggers actions that in their turn produce new events.

4 Conclusion

In this paper, we first described a prototype system that facilitates e-commerce activities and is based on a set of Internet technologies, including XML, Java Servlets, EJBs, and IBM Websphere. Currently, we extend the pro-

totype architecture to include ECA scripts so that, e-commerce applications can be developed and deployed in a customizable and extensible way. This leads to further development of a generic architecture using a Rule Engine to replace the XML Integrator. This new architecture will allow the user to customize e-commerce applications by encoding as ECA scripts specific business transaction logic.

References

[1] IBM XML Tools, http://www.alphaworks.ibm.com/ and http://www.ibm.com/developer/xml/

[2] W.Kou, et al, Demo: E-Commerce Application of XML Application Framework for e-Business, CASCON 1999, November 8-11, 1999.

[3] W.Kou and Y.Yesha, *E_Commerce Technology trends: Challenges and Opportunities*, Midrange Computing, Spring 2000, ISBN 158347-009-3.

[4] Mylopoulos, J., Gal, A., Kontogiannis, K., Stanley, M., "A Generic Integration Architecture for Cooperative Information Systems" *in Proceedings of Co-operative Information Systems'96*, Brussels, Belgium, pp.208-217.

[5] XML Forum http://www.xml.org

[6] H.Ryan et. al eds. *Practical Guide to Client Server Computing* CRC Press, USA 1998.

[7] M. Brodie *Migrating Legacy Systems*, Morgan Kaufman Publishers, 1995.