XML-enabled Workflow Management for E-Services across Heterogeneous Platforms

German Shegalov, Michael Gillmann, Gerhard Weikum
University of the Saarland, Department of Computer Science
P.O.Box 151150, D-66041 Saarbruecken
e-mail: {shegalov,gillmann,weikum}@cs.uni-sb.de
WWW: http://www-dbs.cs.uni-sb.de
phone: +49 681 302 4811

Abstract

Advanced e-services require efficient, flexible, and easy-to-use workflow technology that integrates well with mainstream Internet technologies like XML and Web servers. This paper discusses an XML-enabled architecture for distributed workflow management that is implemented in the latest version of our Mentor-lite prototype system. The key asset of this architecture is an XML mediator that handles the exchange of business and flow control data between workflow and business-object servers on one side and client activities on the other side via XML messages over http. Our implementation of the mediator has made use of Oracle's XSQL servlet. The major benefit of the advocated architecture is that it provides seamless integration of client applications into e-service workflows with scalable efficiency and very little explicit coding, in contrast to an earlier, Java-based, version of our Mentor-lite prototype that required much more code and exhibited potential performance problems.

1 Introduction

1.1 Motivation

Advanced e-services, such as electronic auctions (e.g., Ebay), all-in-one travel planning (e.g., Expedia), automation of real-estate purchase (e.g., Realtor), or computerized court trials with electronic lawyers (still fictituous today), pose both old and new problems to the underlying software infrastructure. The new ones include, for example, the setup of behavioral contracts between services for the composition of value-added, higher-level services (see, e.g., [AFH+99, BCL+00, CIJ+00, Fra99, HLG+00, LBS+99]) or the use of such services from mobile agents (see, e.g., [CZB+99, KSD99, Pap99, Vei99]). Among the old, but still mostly unsolved, problems is the integration or mediation of the business objects, typically running on heterogeneous platforms, as the basic building blocks out of which e-services are composed. The challenge in this lies in making the integration task as simple as possible, so that new e-services can be configured, deployed, and operated at very little cost (i.e., without an expensive data and system administration staff).

A business object (e.g., a purchase order or a mortgage) consists of a collection of data and some limited-scope piece of program logic (ideally both encapsulated and hidden behind an ADT interface). The classical approach for integration is to develop a unified access interface to the heterogeneous data itself through schema reconciliation and explicit data transformations (see, e.g., [ACM90]). However, it can be taken for granted that this kind of relatively tight integration requires intensive human efforts, and quite a few (overly) ambitious projects towards enterprise-wide data models are known as expensive failures. A much cheaper and more viable approach for integrating business objects is to wrap objects with explicit business-logic interfaces and "simply" orchestrate the control and data flow between objects at the level of an object-flow or work-flow mediator. Object-oriented middleware like CORBA, DCOM, or EJB has aimed at this kind of integration, but is more

focused on wrapping the components and routing primitive requests rather than the actual mediation. The technology that really has all the ingredients to solve this integration issue is workflow management (see, e.g., [GSC+99, Ley95]). A workflow consists of a set of activities, automated or intellectual/interactive ones, with explicitly specified and system-enforcable control and data flow. Each activity can in turn be viewed as invoking a method on a business object.

State-of-the-art workflow management systems, in conjunction with the underlying middleware such as CORBA, are indeed capable of integrating business objects for setting up a new e-service in amazingly short time and thus with impressively little cost (e.g., virtually no code-writing or all code automatically generated from easily constructed high-level specifications). However, the pace of the ongoing trends towards functionally richer e-services on the Internet has been so fast that the vendors of workflow management systems have had hardly any time to prepare their products for these new settings. So probably no workflow product is ready for deployment on Internet-wide, arbitrarily heterogeneous platforms on a short-notice basis. The issues of how to leverage the latest developments on XML-centered technologies and how to combine the best of several worlds into an easy-to-use and mostly administration-free infrastructure are widely open.

1.2 Contribution

This paper proposes an architecture that leverages XML technology for Internet-wide workflow management within advanced e-services. The key concept is to "marshall" all activity calls into special kinds of XML documents and to handle the flow between the workflow engine and the activities by some kind of XML mediator using http as the primitive transport protocol. Thus, workflow engines as well as activities and the corresponding business objects can reside on their native platforms without any special measures, and the XML mediator enables the cross-talk among all these components with all data exchanges in the form of XML documents. None of the involved parties, clients, business-object servers, or workflow servers, needs to know any details about the other parties. Clients need no software other than an Internet browser, yet handle interactions with workflows and business objects much more efficiently than with a full-blown Java implementation.

The paper presents this architecture and contrasts it with a more traditional setup that builds more heavily on CORBA (or equivalently EJB) and Java. We believe that this XML mediator approach will be beneficial as a general infrastructure and is not necessarily tied to our current context of workflow management. Its salient properties are the following:

- It builds on a rather lean software infrastructure that is virtually ubiquitous anyway: an Internet browser with simple XML/XSL support on the client side, an http server with efficient support for servlets at the mediator level, and standard SQL via JDBC and the most basic object-oriented middleware services (either CORBA or EJB) to interoperate with the various business-object and workflow servers.
- As a consequence, the entire infrastructure is very easy to set up and administer. In particular there are no extra software installations on clients and no special requirements on the mediator's http server; so much of the usual headache with software distribution, compatibility of software releases, and general maintenance is eliminated (or reduced to what is necessary for a ubiquitous standard infrastructure anyway).
- The mediator as a "middle man" bundles the traffic between the workflow and business-object servers on one side and (automatic as well as interactive) applications on the other side. So the mediator can reuse permanently established threads and connections with the backend servers, and generally enjoys all the scalability benefits that have made three-tier architectures so prevalent on the Web. Furthermore, the computing resources of the mediator can be exploited to alleviate the load on the client or business-object server side, in adaptation to where the bottlenecks are. In particular, the mediator can push necessary transformations on the XML data that is routed between business objects and clients either to the client, making more use of XSL and browser-embedded scripts, or to the business-object servers, making more use of advanced SQL or native XML handling capabilities, or it can choose to do the bulk of this work by itself.

The proposed architecture has been fully prototyped in the latest version of our Mentor-lite distributed workflow management system [GWS+00, WGR+00]. The XML mediator itself is implemented using Oracle's

XSQL servlet, run under the Apache Web application server, with a small number of very compact so-called custom XSQL action handlers. All client functionality is embedded in an Internet browser (specifically IE5) and implemented with a fairly small amount of XSL and DHTML code.

1.3 Outline

The rest of the paper is organized as follows. Section 2 mentions related work. Section 3 discusses system architectures for Internet-based workflow, including the XML-enabled architecture advocated here. Section 4 discusses more specifically how the control and data flow among workflow activities is handled by the XML mediator. Section 5 presents a simple e-commerce scenario that we have implemented on our prototype system.

2 Related Work

Architectural issues of workflow management systems have been intensively explored in research and development (see, e.g., [DKO+98, GHS95, JB96, LR99]). Much emphasis has been put on scalability and robustness to ensure industrial-strength service. Today's most advanced products may indeed claim that they are ready for mission-critical, enterprise-wide use in terms of performance and availability. However, much of this virtue comes at the cost of a fairly large system footprint (e.g., memory requirements of the workflow system) and careful administration.

Combining workflow technology with the Internet has been successfully addressed with regard to browser-based user interfaces (e.g., for worklists) and transactional as well as "transport level" protocols (see, e.g., [Moh99, SDD+97]), but there has been very little work on a deeper integration of Web and workflow technologies and their software infrastructures. In fact, the latent workflow functionality in most of today's e-commerce applications has mostly been implemented in an ad-hoc manner; typically the workflow state and context is maintained by a collection of small servlets (e.g., Active Server Pages in IIS or PHP scripts under Apache).

XML technology has been detected by the workflow community only recently. At this stage, the work in this direction appears to be limited to casting workflow specifications and system interface descriptions into XML format (see, e.g., a draft of the Workflow Management Coalition [WfMC00]). This work is on the right track, but there is hardly any implementation work along these lines (one exception that we are aware of being the recent work of [CHD+99]). XML as a container for remote method invocation, with http as an underlying "transport" protocol, is intensively pursued in Microsoft's SOAP protocol (Simple Object Access Protocol) and the corresponding efforts towards a W3C standard [SOAP]. However, SOAP aims to be a lowest common denominator among all classes of Internet applications, whereas our work focuses on richer, workflow-style, advanced e-services.

3 System Architecture

In this section we discuss different system architectures for Internet-based workflow management. We begin, in Subsection 3.1, with the reference architecture of the Workflow Management Coalition [WfMC], an industry consortium that aims to standardize workflow-system interfaces for interoperability; this serves as a baseline against which we can compare the architectures of real systems. In Subsection 3.2 we present the architecture of our prototype system, coined Mentor-lite [MWG+99], as it looked a year ago. As this version of our prototype made extensive use of Java and CORBA services, we refer to it as the Java-based architecture. Only recently we have re-architected the Mentor-lite system, and the latest version is centered around XML. This is the architecture that we believe is most suitable for advanced e-services; it will be introduced in Subsection 3.3.

3.1 Reference Architecture

Workflow specification languages range from Petri-net-like or statechart-style high-level visual languages or specific types of (modal) logic all the way to scripting or simply (unstructured) collections of (ECA) rules. A workflow specification is often derived from a (business) *process modeling and definition tool* such as Aris Toolset. Upon initiation by a user a workflow specification is instantiated and interpreted by a *workflow engine*. One or more engines (e.g., on an SMP computer) form the *workflow enactment service*. During its execution a workflow spawns *activities*. These activities correspond to either *client applications* or other *invoked applications*; typically the first correspond to interactive activities such as intellectual decision making (possibly using tools such as spreadsheets) and the second to automated activities such as host applications. Workflows can spawn entire subworkflows that may be controlled by other, "external" workflow engines (running software from different vendors on servers that belong to different organizations). Finally, for analyzing workflow usage patterns and providing feedback to the business process re-engineering lifecycle, *administration and monitoring tools* are needed.

All these components and their interactions are cast into a nice framework known as the reference model of the Workflow Management Coalition [WfMC], depicted in Figure 1. Most importantly, the WfMC has also issued standards for the five relevant APIs in this architecture. In [WfMC98] they discuss the impact of modern Internet technologies on the various APIs of the reference architecture. Obviously and most importantly, API2 should consider the fact that Internet browsers are powerful tools and already provide capabilities to embed a rich suite of client applications through plug-ins, ActiveX, Java applets, etc. As for API3 the WfMC brings up for consideration the CORBA-oriented IIOP (Internet Inter-ORB Protocol) and the services of EJB (Enterprise Java Beans) for invoking appropriately wrapped host applications. Finally, the WfMC also discusses the potential benefit of casting all workflow APIs into Java for Internet-enabling. Although these considerations by the WfMC are very useful in sorting out the various issues and establishing a common framework, they stay at an abstract level and are of very limited help in making design decisions for a concrete implementation. Most recently, the WfMC has also issued a document on Wf-XML [WfMC00] that describes how calls and parameters of the various APIs can be cast into XML format. However, how these XML messages should actually be embedded into the architecture of a real system is left widely open.

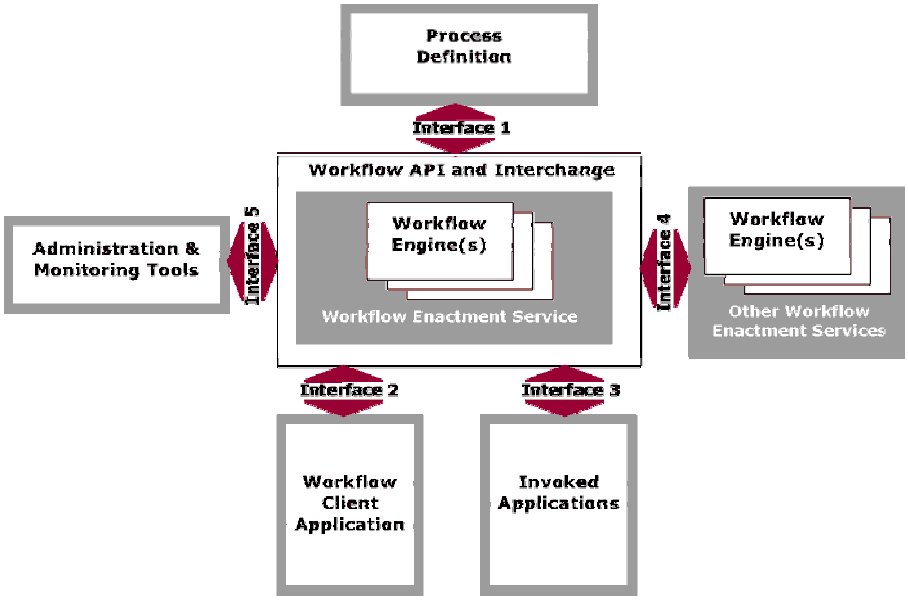

Figure 1 : Reference architecture of the WfMC

3.2 Java-based Architecture

Following the proliferation of Java, our first implementation of the Mentor-lite prototype [MWG+99] has made intensive use of Java on the client side and CORBA as general middleware. Our workflow engine, an interpreter for a specifically designed statechart dialect [WW97], is run as a CORBA application server, coined DSIserver (for Distributed Statechart Interpreter), or as a collection of such servers on different computers with support for decentralized execution of subworkflows [GWW+00, MWW+98a, MWW+98b]. More specifically, we have used Iona's Orbix product [IONA], one of the few industrial-strength and CORBA-compliant request brokers. The DSIserver does itself make use of an Oracle8i database for reliably tracking the workflow state and context, and also for managing organizational and history data that is relevant for worklist policies (i.e., role resolution). This database also holds the current states of worklists and their work items.

CORBA technology is used for the API3 to invoke host applications (i.e., these applications must be wrapped with an IDL interface), and also for the API4 to other workflow engines. The latter is exploited already within Mentor-lite when workflows span multiple, organizationally separated sites each of which autonomously runs a DSIserver for the subworkflows that it is responsible for. In this case the cross-engine communication takes place via transactional, reliable message queues that we have ourselves implemented as a specific queue server using Orbix OTS (Object Transaction Service) as a 2PC coordinator. This setup can also handle data exchanges among heterogeneous workflow servers, running engines from different vendors. However, our implementation used home-grown message formats (e.g., for signaling changes of the workflow state and context to another engine). Casting these messages into XML would allow us to coordinate workflows across heterogeneous platforms.

As for API2 between the workflow engine and the client applications and also API5 with regard to administration tools, we had chosen to make intensive use of Java applets. Worklists and their work items are presented to the user by applets embedded in the user's Internet browser. These applets are dynamically loaded from a trusted Web server. The initial applet is launched and contacts the DSIserver for work-to-do, which is implemented as a http call and handled via the IIOP-based OrbixWeb service (which needs to be installed on the client side). Once running, all applets directly access the Oracle8i database with the relevant worklist data using JDBC for performance reasons, as opposed to always having an indirection through the DSIserver (i.e., the workflow engine). When the user selects a specific work item from her worklist, the corresponding client application is launched as another applet that may itself invoke standard tools (e.g., a spreadsheet program via ActiveX). This overall architecture of our first Mentor-lite version is illustrated in Figure 2.

Based on this Java-centric version of Mentor-lite we built a mid-sized prototype application for student enrollments, exams, etc. within our university department. The architecture proved to be viable, but we also observed a number of performance-critical issues:

1. Loading the Java applets from a trusted server into the client turned out to be time-consuming even within an Intranet (i.e., over high-speed LAN technology, but involving the usual software overhead). As these applets access resources outside the browser's sandbox by making JDBC calls, remote loading from a trusted server could not be easily eliminated. In addition, we also learned that portability and "zero-admin" installation of Java applets that include JDBC, CORBA, ActiveX, etc. calls is all but self-guaranteed because of platform-specific resource restrictions of the browser sandbox.
2. The JDBC calls themselves were relatively expensive, too. The delays due to a good number of message roundtrips were sometimes user-noticeable. In particular, the setup of a JDBC session is relatively expensive in terms of message cost. Also, as JDBC is limited to dynamic SQL (i.e., cannot be pre-compiled), the run-time overhead at the database server was also significant.
3. With every client establishing a JDBC session with the database server, scalability may be a potential problem. Note, however, that this approach was still signficantly more efficient than the alternative of involving the Orbix-based DSIserver on each and every interaction with the worklist.
4. For cross-organizational workflows over heterogeneous workflow engines, clients would have to maintain even more sessions with a suite of underlying servers. With session setup being a fairly expensive part of the protocols, even powerful clients may exhibit delays that could slow down user interactions.

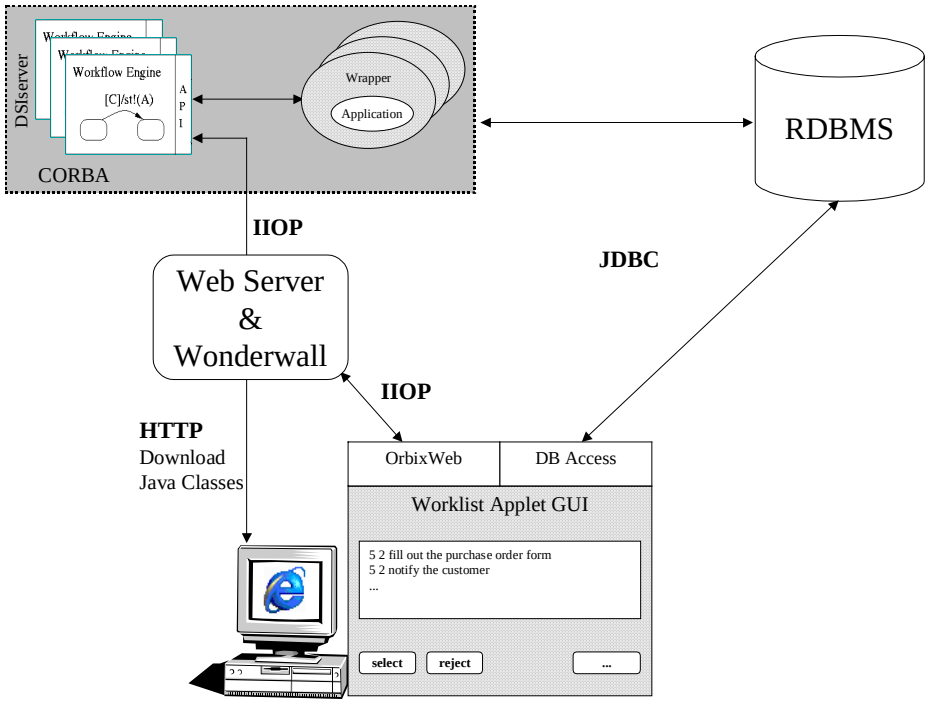

Figure 2 : Architecture of the Java-based Mentor-lite version

3.3 XML-based Architecture

Based on the lessons learned with the Java-based architecture, we redesigned the interfaces to client applications (API2) and invoked applications (API3) and modified the affected components of the Mentor-lite prototype system, leading to what we refer to as the XML-based version of Mentor-lite.

In essence, we modified the messages that are received and sent by clients and invoked applications so that they are in XML format, using a set of Mentor-lite-specific XML tags. Messages now contain the activity name and workflow id as an XML element, and further XML elements for input and output parameters. For readability and easier processing we discriminated parameters into two classes:

- *Business data* are input or output parameters that influence the persistent business objects on which the activity's application operates. For example, when invoking an activity to receive a new order, the customer id, id of the ordered product, and the ordered quantity are business data.
- *Flow control data* are input or output parameters that "merely" serve to drive the control (and data) flow among activities in the workflow, but have no long-term impact on persistent business objects beyond the scope of the current workflow. This category especially includes return codes of activities.

We re-implemented the client software so that it makes use of the browser's capabilities for local XML processing (using IE5 in our implementation). The client can filter and sort data through XSL stylesheets, which is exploited on worklists, and DHTML (Dynamic HTML) is used for presenting XML data in a user-friendly GUI. This modification led to a drastic reduction of the client code and eliminated performance problem 1 of the earlier Java-based architecture (see Section 3.2). The client code for the student enrollment and exams application mentioned in Section 3.2 was reduced to a few hundred lines of JavaScript/JScript and XSL stylesheets.

To address the other performance problems mentioned in Section 3.2 we introduced an *XML mediator* as a "middle man" between the clients and the workflow engines as well as the Oracle8i database server. This mediator was designed as a servlet that can run in any http server. Its purpose is to feed XML data to the client

activities and invoked applications and collect their XML output, while being able to communicate with the backend servers much more efficiently. In particular, the mediator servlet can maintain a moderate number of permanently open JDBC sessions with the Oracle8i database server, which eliminates performance problems 2 and 3 to a large extent (see Section 3.2).

As for the mediator implementation we massively benefitted from the recent release of Oracle's XSQL servlet [XSQL] (which was still in Beta status at the time when we did our implementation). This service, which is implemented in Java and can run in any standard http/servlet server (e.g., Apache, which is used in our prototype), mediates between the SQL and the XML worlds by converting the results of SQL statements into XML and generating SQL calls to store incoming XML data via JDBC. Most importantly, XSQL could be easily extended through so-called custom XSQL action handlers to accomodate functions that are specific to our workflow environment. We will present details of this customized extension in Sections 4 and 5.

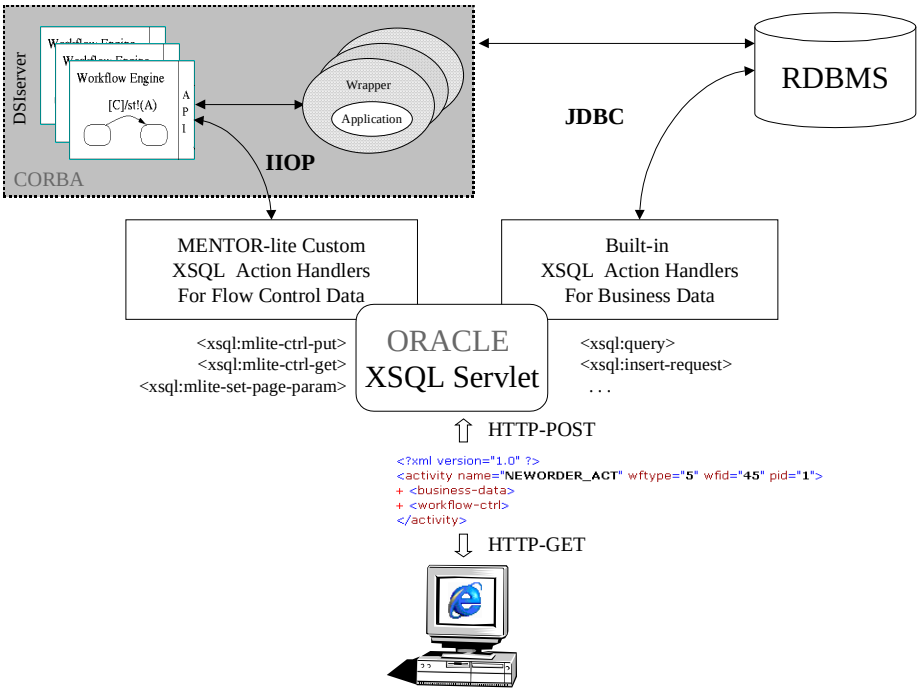

Figure 3 : Architecture of the XML-based Mentor-lite version

The XML mediator is also the ideal middle man in a cross-organizational workflow with highly heterogeneous server platforms, as it encapsulates the servers with regard to the clients and can maintain a moderate number of sessions with heterogeneous workflow or business-object servers with reasonable efficiency. Thus, problem 4 mentioned in Section 3.2 is also, to a large extent, rectified in the new architecture.

The overall XML-based architecture of Mentor-lite is illustrated in Figure 3.

4 The XML Mediator

4.1 Conceptual Overview

The XML mediator provides a high-level abstraction for WFMS interoperability in the context of global (i.e., cross-enterprise) workflow management. It offers an Internet-based, worldwide accessible interface to invoke workflows or activities and pass results to the proper destinations. By viewing an activity as an abstraction of either an invoked application or a subworkflow the mediator enables seamless integration of all

kinds of business activities. As underlying IT infrastructure we consider workflow, application, and business-object servers, where the latter are typically database servers. Our goal is to hide all infrastructure details (location of servers etc.) behind a simple XML interface.

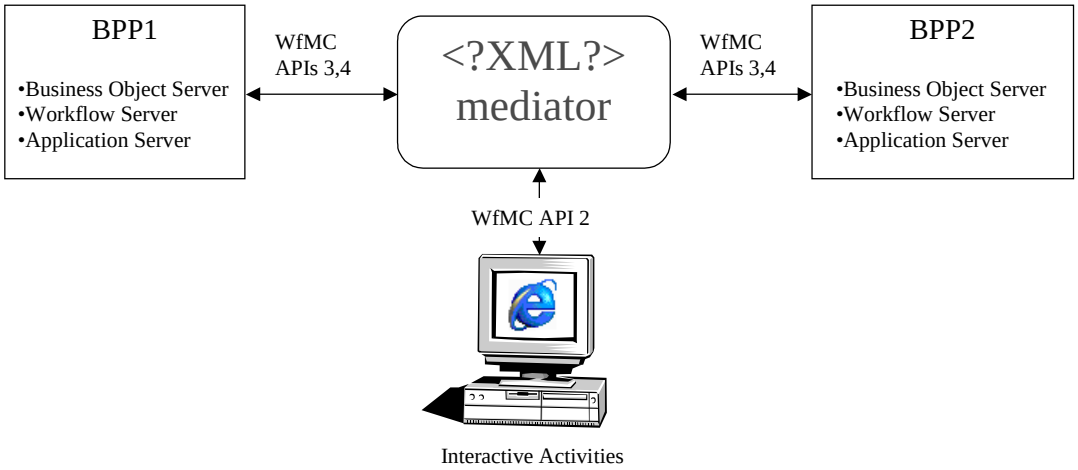

Figure 4 : Role of the XML mediator

Consider a scenario, depicted in Figure 4, with two enterprises involved in a global business process, i.e., an e-service such as providing personalized multimedia newspapers to subscribers. Both business process participants, BPP1, which could be the business portal for this e-service, and BPP2, which could be the content provider, have their own specific IT solutions. The XML mediator facilitates the interoperation between these two information systems, thus making global workflows feasible (e.g., for setting up the service of a new subscriber). There may also be interactive activities, which are introduced solely for the purpose of the global workflow. The XML mediator hides implementation details (underlying middleware, firewalls etc.) of BPP1 and BPP2 to each other party and to the interactive activity. Special XML messages that contain only business and flow control data are exchanged between the various parties via the XML mediator. The mediator is responsible for the message delivery according to the receiver's identification and the tags in the XML message. In our prototype, for example, different tags are used to identify business vs. flow control data, with the latter being delivered to the workflow engine and the business data sent to an activity or, upon completion of the activity, back to a database server. Note that an activity is itself an abstraction that comprises interactive activities (the case on which we mostly focus), automated activities that invoke applications, or activities that encapsulate a subworkflow running on a different workflow engine.

We identify activities, which may be subworkflows by a triple <workflow-type, workflow-id, process-id>. XML messages that are sent to an activity to invoke the activitiy (and whatever application or subworkflow is behind the activity) are structured as follows:

```
<?xml version='1.0' ?>
<activity name='some name' wftype='x' wfid='y' pid='z'>
  <business-data>
    <relation1><row1><attr1>...</attr2></row1> ... </relation1>
    ...
  </business-data>
  <workflow-ctrl>
    <variable name='var-name1' value='var-value1'/>
    ...
  </workflow-ctrl>
</activity>
```

Here we assume that business objects typically reside in relational databases, but extensions to incorporate object-oriented or object-relational databases would be straightforward. When an activity completes, it sends an analogously structured XML message with its output to the mediator, which then parses the message and derives the necessary follow-up actions like storing new or modified business objects and sending flow control data to a workflow engine.

4.2 Interfaces

Upon the start of an activity, the client's Web browser obtains its input data in XML format by an http get call. Analogously, results from the activity are returned by an http post call. The corresponding DSIserver to which the output should eventually be delivered is referenced by the triple <workflow type, workflow ID, process ID> which is automatically resolved by the XML mediator using the CORBA name service. So the application that implements the activity on the client side need not know details about where input data comes from and how result data is handled further on.

The core of the XML mediator is implemented using Oracle's XSQL servlet. This Java servlet serves to combine results of SQL queries into XML documents or to extract SQL updates from such documents. A strong feature of the XSQL servlet is the option for adding "custom XSQL action handlers" that can be invoked by the XSQL servlet based on special XML elements, marked by the prefix *xsql*, in the XSQL page. Both, custom and built-in action handlers conform to the Java interface shown in Figure 5. The function *init* (line 2) serves to initialize the action handler, typically by reading the attributes of the action element. The *handleAction* function (line 3) executes the action itself. The parameter *rootNode* refers to the node that will be inserted in place of the action element in the resulting XML document. For each appearance of an XSQL action element in the XSQL page the XSQL servlet creates a new instance of the corresponding action handler class, and invokes the method *init* and *handleAction* on the recently created instance subsequently [XSQL].

```
1   public interface XSQLActionHandler {
2     void init(XSQLPageRequest env, Element actionElement) ... ;
3     void handleAction(Node rootNode) ... ;
4   }
```

Figure 5 : Generic Java interface of the XSQL action handlers

Custom action handlers are needed in the context of workflow management to handle the flow control and business data that are exchanged between an activity and the workflow engine. Table 1 shows three basic XML elements and corresponding action handlers for reading and manipulating control flow variables, i.e., variables that are required by the workflow engine for control flow decisions (e.g., return codes).

action element	**action handler class**	**action**
<xsql:mlite-ctrl-get>	mlite.controlFlowGetHandler	to read flow data
<xsql:mlite-ctrl-put>	mlite.controlFlowPutHandler	to write flow data
<xsql:mlite-set-page-param>	mlite.setPageParamHandler	to read flow data to be set as a parameter at the page-level

Table 1 : Mentor-lite specific custom XSQL action handlers

To keep the implementation of the custom action handlers simple, we have designed a class hierarchy shown in Figure 6. All action handler classes are derived from the basic class *mliteHelper* that includes generic functions like communication interfaces (e.g., requests to the CORBA name service). The class *mliteHelper*

itself extends the *XSQLActionHandlerImpl* class, a base implementation of XSQL action handlers that includes a set of useful helper methods (e.g., for error handling and monitoring). The *XSQLActionHandlerImpl* class is part of the runtime library by Oracle.

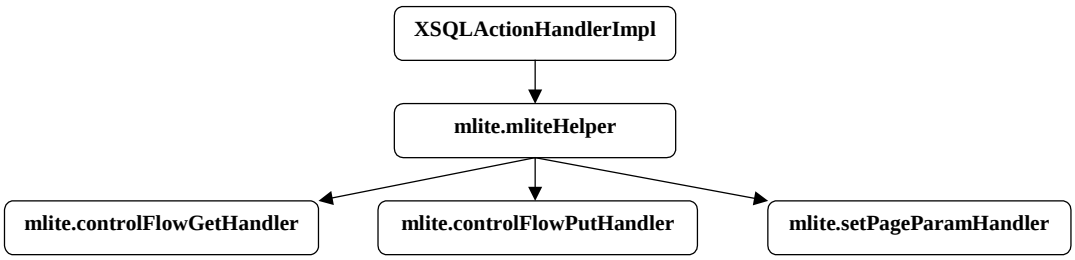

Figure 6 : Class hierarchy of the action handlers

4.3 Implementation

Figure 7 shows the implementation of the *mliteHelper* class that extends the base implementation of the *XSQLActionHandlerImpl* class. It provides a simple API to identify the workflow engine responsible for the current workflow instance and the primary error handling that is necessary for using the CORBA name service.

```
 1   public abstract class mliteHelper extends  XSQLActionHandlerImpl {
 2       private static  NamingContext rootContext;
 3       private static org.omg.CORBA.ORB orb;
 4       static {
 5           System.err.print("initializing CORBA interface ... ");
 6           orb = ORB.init((String[]) null, null);
 7           System.err.println("done");
 8           System.err.print("initializing NamingService ... ");
 9           try {
10               org.omg.CORBA.Object ns =
11                           orb.resolve_initial_references("NameService");
12               rootContext = NamingContextHelper.narrow(ns);
13           } catch(Exception e) {
14               System.err.println(e.getMessage());
15           }
16           System.err.println("done");
17       }
18       public dsiserver getWfEngine(String engineName) throws Exception {
19           System.err.print("Search for the engine " + engineName + " ... ");
20           NameComponent[] name = new NameComponent[1];
21           name[0] = new NameComponent(engineName, null);
22           org.omg.CORBA.Object wfe = rootContext.resolve(name);
23           System.err.println("done");
24           return mlite.dsiserverHelper.narrow(wfe);
25       }
26       public void insertException(org.w3c.dom.Node n, Exception e) {
27           this.reportError(n, e.toString());
28           e.printStackTrace();
29       }
30   }
```

Figure 7 : Implementation of action handler *mliteHelper*

The lines 4 to 17 initialize the CORBA interface and connect to the name service. The function *getWfEngine* (lines 18 to 25) determines the workflow engine that is responsible for the execution of the current workflow instance. So the application programs that implement the workflow activities do not need to know anything about the location of the engine or its distribution over the network. The function *insertException* (lines 26 to 29) supports exception handling.

```
/* error handling code isn't shown here */
 1   public class controlFlowPutHandler extends mliteHelper {
 2       public void handleAction( Node rootNode ) throws SQLException {
 3           Document xmlDoc  = this.getPageRequest().getPostedDocument();
 4           Element xmlRoot  = xmlDoc.getDocumentElement();
 5           String wftype    = xmlRoot.getAttribute("wftype");
 6           String wfid      = xmlRoot.getAttribute("wfid");
 7           String pid       = xmlRoot.getAttribute("pid");
 8           NodeList wfCtrlBlock =
 9                    xmlRoot.getElementsByTagName("workflow-ctrl");
10           NodeList variables =
11             ((Element) wfCtrlBlock.item(0)).getElementsByTagName("variable");
12           int varcount = variables.getLength();
13           String engine_name = "dsiserver"   +
14              wftype + '_' +
15              wfid   + '_' +
16              pid;
17           try {
18               dsiserver wfengine = this.getWfEngine(engine_name);
19               Element curr_elem;
20               String varval;
21               for(int i=0; i < varcount; i++) {
22                   curr_elem = (Element) variables.item(i);
23                   varval =
24                       curr_elem.getAttribute("name") +
25                       '=' +
26                       curr_elem.getAttribute("value");
27                   wfengine.put(varval, "");
28               }
29               wfengine._release();
30               this.reportStatus(rootNode,
31                                 "result",
32                                  varcount + " variables inserted");
33           } catch(Exception e) {
34               insertException(rootNode,e);
35           };
36       }
37   }
```

Figure 8 : Implementation of action handler *controlFlowPutHandler*

As an example of the implementation of a workflow-specific action handler, Figure 8 shows the Java code for the *controlFlowPutHandler* class that is associated with the action element <*xsql:mlite-ctrl-put*> and will be automatically called by the XSQL servlet when processing the XML output of an activity. The *controlFlowPutHandler* class overwrites the method *handleAction* of the base implementation. When *handleAction* is invoked, we obtain a reference to the posted XML document as an XML DOM object by calling the method *getPostedDocument* of the interface *XSQLPageRequest* in line 3. Subsequently we are able to manipulate or read nodes of the posted XML document in lines 4 to 7. First we extract the attributes *wftype*, *wfid*, and *pid* from the root node *activity*, i.e., the responsible DSIserver can be identified by calling the method

getWfEngine of the base class mliteHelper. The list of the control flow variables is obtained by calling the *getElementsByTagName* (lines 8 to 11) on the workflow-ctrl DOM element object. After the connection to the workflow engine is established (line 18), flow control data can be transferred to the workflow engine by calling *wfengine.put* method (lines 21 to 28).

5 A Simple Case Study

In this section, we present the implementation of a simplified e-commerce scenario as an example of a workflow application. This case study serves as a proof of concept for our mediator approach. In addition, it shows that the implementation of new workflow applications for e-services can be carried out in a straightforward manner in very short time and thus with very low cost. In the example, we focus on the interface between the workflow engines and the applications invoked for interactive activities.

The workflow is based on the TPC-C order-entry benchmark for transaction systems [TPC], with the key difference that we combine multiple transaction types into a workflow and further enhance the functionality (see [GMW+99] for a full description of this workflow).

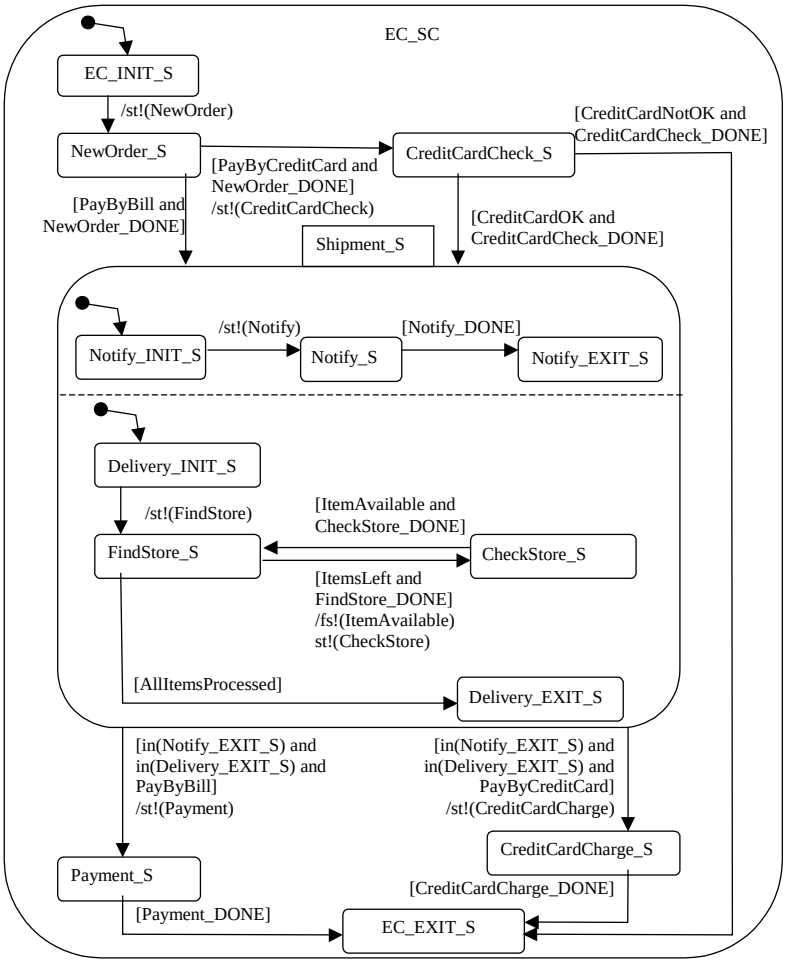

Figure 9 : State chart of the *electronic commerce (EC)* workflow example

Figures 9 and 10 show the workflow specification as a state and activity chart, a formalism [Har87, HG97] that has been adopted for the behavioral dimension of the UML industry standard and is used in our prototype

system Mentor-lite [WW97, MWW+98a]. Each state in Figure 9 corresponds to an activity in Figure 10 or one (or multiple, parallel) subworkflow(s), except for initial and final states. We assume that for every activity *act* the condition *act*_DONE is set to true when *act* is finished.

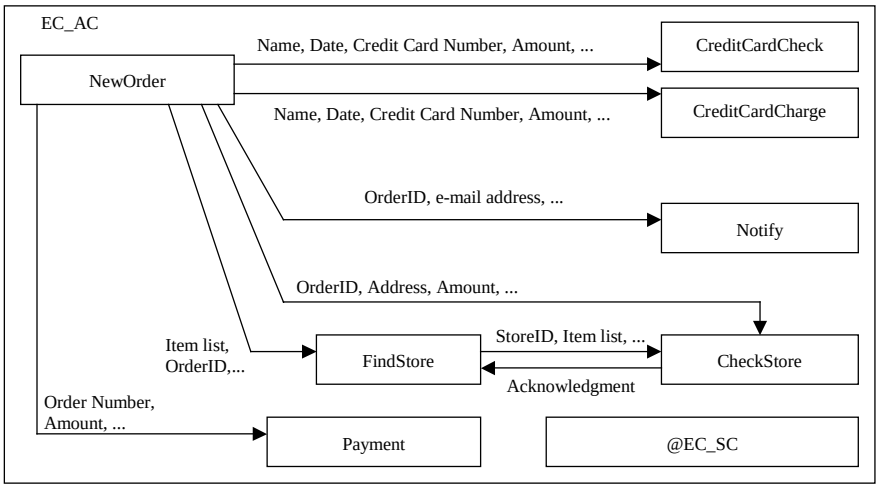

Figure 10 : Activity chart of the *electronic commerce (EC)* workflow example

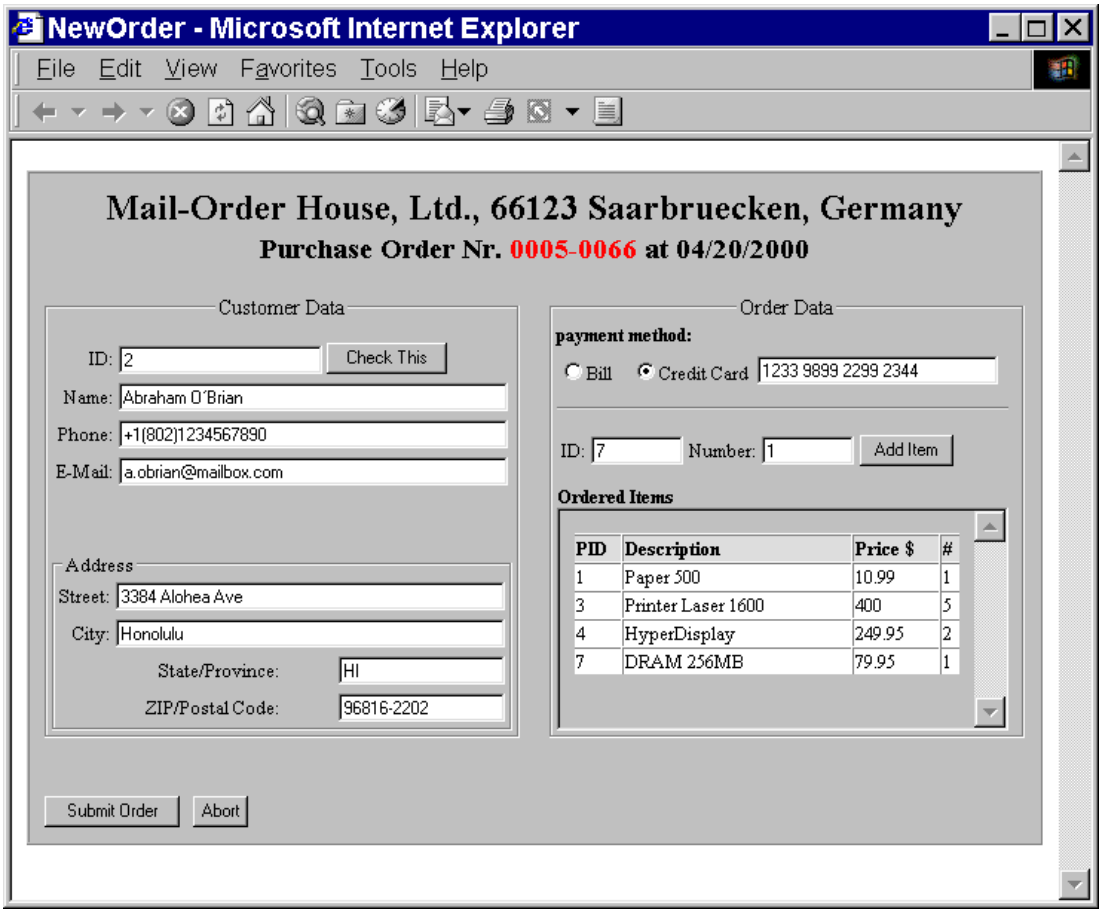

Figure 11 : Output of the DHTML application for activity *NewOrder*

The workflow proceeds as follows. Initially, the *NewOrder* activity is started. After the termination of *NewOrder*, the control flow branches. If the customer wants to pay by credit card, the condition *PayByCreditCard* is set and the *CreditCardCheck* activity checks the validity of the credit card. If there are problems with the credit card, the workflow is terminated. Otherwise the shipment, represented by the nested top-level state *Shipment_S*, is initiated spawning two orthogonal/parallel subworkflows. The first subworkflow has only one activity that sends a notification mail. The second subworkflow (sequentially) invokes for each ordered item an activity that identifies a store from which the item could be shipped. Then, a second activity instructs the store to deliver the item and waits for an acknowledgement. The two activities *FindStore* and *CheckStore* are repeated within a loop over all ordered items. After the termination of both subworkflows, the control flow is synchronized, and branches again depending on the mode of payment. The workflow terminates in the state *EP_EXIT_S*.

In the following, we look closer at the *NewOrder* activity to illustrate the role of the XML mediator in the activity handling. This activity is implemented as a Dynamic HTML (DHTML) application (see Figure 11). In general there are two approaches to handle XML on the client side using Microsoft IE 5.0. First, the XML data can be transformed by the stylesheet engine of the browser into DHTML after downloading XML and XSL. Alternatively, a client application can be implemented in DHTML directly. In the latter case so-called "XML data islands" [XML] can be defined within a DHTML document, in order to separate the input data from the presentation. An XML data island can easily be bound ("Data Binding" [XML]) to an HTML element as a dynamic source object (DSO) to be visualized within the DHTML document. The main advantage of using this Data Binding is that the HTML element will be updated automatically if the corresponding DSO has changed. Once initialized in the browser, the DHTML document for the *NewOrder* activity contains an empty XML data island <xml id=”dsoCustomer”></xml>. The <xml id …> tag is a special HTML tag for defining XML data islands. For convenience the user may simply input the customer id and click on the “*Check This*” button. On this event the function *checkCustomer()* (Figure 12) will be executed, which loads the content of the XSQL page *customer.xsql* (Figure 13), processed by XSQL servlet, into the XML DOM object *dsoCustomer* (line 4). Form fields like "Phone" are automatically filled with the content of the <*phone*> tag of *dsoCustomer* (line 6 in Figure 14) by specifying <input size="50" name=”phn” datasrc="#dsoCustomer" datafld="phone"> in the DHTML script. The attributes *datasrc* and *datafld* refer to the DSO object and the field of interest in this object. All other form fields are filled analogously (lines 3 to 14).

```
1   function checkCustomer() {
2     dsoCustomer.async = false;
3     url = "customer.xsql?customerID=" + customer.ID.value;
4     dsoCustomer.load(url);
5   }
```

Figure 12 : JScript function for fetching the customer data according to the customer ID (part of the DHTML code for activity *NewOrder*)

```
 1   <?xml version="1.0" ?>
 2   <customer connection="mlite-demo">
 3     <xsql:query xmlns:xsql="urn:oracle-xsql"
 4          rowset-element="" id-attribute=""
 5          row-element="" tag-case="lower">
 6       SELECT *
 7       FROM CUSTOMERS
 8       WHERE id={@customerID}
 9     </xsql:query>
10   </customer>
```

Figure 13 : XSQL page customer.xsql

```
 1   <?xml version="1.0" ?>
 2   <customer>
 3      <id>2</id>
 4      <name>Abraham O´Brian</name>
 5      <mail>a.obrian@mailbox.com</mail>
 6      <phone>+1(802)1234567890</phone>
 7      <address>
 8         <street>3384 Alohea Ave</street>
 9         <city>Honolulu</city>
10         <state>HI</state>
11         <zip>96816-2202</zip>
12      </address>
13      <balance>0</balance>
14      <discount>0</discount>
15   </customer>
```

Figure 14 : Output produced by XSQL servlet after processing *customer.xsql*

Unlike customer data, items can be added to the "shopping cart" several times. To avoid repeatedly checking item Ids by calling the XML mediator, we prefetch relevant parts of the product catalog into the browser by defining an xml data island <xml id="dsoProducts" src="products.xsql"></xml>. The XSQL page *products.xsql* and an example of the resulting document are shown in Figures 15 and 16. Now suppose the user inputs id=7 and quantity=1 and clicks on the "*Add Item*" button. In order to check the details of this item we call *dsoProducts.selectSingleNode("//product[id=7]")*. All manipulations on the DHTML page such as adding items are processed locally without involving any servers or the XML mediator.

```
1   <?xml version='1.0' ?>
2   <products connection='mlite-demo'>
3     <xsql:query xmlns:xsql="urn:oracle-xsql" …>
4       select * from products
5     </xsql:query>
6   </products>
```

Figure 15 : XSQL page *products.xsql*

```
 1   <?xml version="1.0"?>
 2   <products>
 3      <product>
 4         <id>2</id>
 5         <description>HDD 20MB</description>
 6         <price>199.95</price>
 7         <store>Detroyt-Store</store>
 8         <supply>100</supply>
 9      </product>
10      <product>
11         <id>3</id>
12         <description>Printer Laser 1600</description>
13         <price>400</price>
14         <store>Boston-Store</store>
15         <supply>310</supply>
16      </product>
      …
     </products>
```

Figure 16 : Resulting content of the XML data island

When the *NewOrder* activity is finished, i.e., the user has entered all ordered items and pushed the submit button, an XML document that includes both the business and flow control data is sent to the XSQL servlet by a http post call. An example for such an output is given in Figure 17. The structure of the output is generic and the same for all activities. The <*activity*> element in line 2 specifies the activity and the corresponding workflow instance followed by a block of business data (lines 3 to 18) and a block of flow control data (lines 19 to 23).

```
 1   <?xml version="1.0"?>
 2   <activity name="NewOrder" wftype="5" wfid="45" pid="1">
 3      <business-data>
 4         <order>
 5            <row>
 6               <order_id>1</order_id>
 7               <customer_id>3</customer_id>
 8               <prod_id>100223</prod_id>
 9               <count>12</count>
10            </row>
11            <row>
12               <order_id>1</order_id>
13               <customer_id>3</customer_id>
14               <prod_id>100002</prod_id>
15               <count>1</count>
16            </row>
17         </order>
18      </business-data>
19      <workflow-ctrl>
20         <variable name="PayByCreditCard" value="1"/>
21         <variable name="PayByBill" value="0"/>
22         <variable name="NEWORDER_OK" value="1"/>
23      </workflow-ctrl>
24   </activity>
```

Figure 17 : XML output of the activity *NewOrder*

The XSQL page that receives the document is shown in Figure 18. The <xsql:insert-request> action element (lines 3 to 5), one of the built-in action elements from Oracle, refers to an XSQL action handler that stores the business data in the specified table. The action element <*xsql:mlite-ctrl-put*> (line 6) is assigned to the action handler *controlFlowPutHandler* that transfers the flow control data to the DSIserver, i.e., the workflow engine, as described in Section 4.

```
1   <?xml version="1.0" encoding="iso-8859-1"?>
2   <activity xmlns:xsql="urn:oracle-xsql" connection="mlite-demo">
3      <xsql:insert-request
4            tag-case="lower"
5            table="orders"/>
6      <xsql:mlite-ctrl-put/>
7   </activity>
```

Figure 18 : NewOrder.xsql for http post

The bottom line of this walk-through of our example scenario is that surprisingly little code is needed to build a simple but not unrealistic e-service application.

6 Concluding Remarks

In this paper we have described an XML-enabled architecture for distributed workflow management on the Internet. The architecture is fully implemented in our Mentor-lite prototype system. In comparison to our earlier, Java-centric, implementation, we have achieved both substantial performance improvements and also a drastic reduction in the coding that is necessary for building workflow applications. Especially for the latter reason we believe that this architecture, that builds on ubiquitous standard infrastructure enhanced by an XML mediator, is particularly well geared for easy, fast, and inexpensive setup of advanced e-services on the Internet.

To this end, it is crucial that clients do not need any specific software installations or setups, as it would be a nightmare to make millions of potential clients ready for e-service usage and maintain their local software environment as services undergo evolution. In this regard, Java applets, upon which our earlier prototype relied, are much less ubiqituous than what marketing makes us believe, once these applets include non-trivial resource manipulation such as IIOP calls. Our current implementation, as presented in this paper, avoids these complications by delegating these non-trivial resource accesses to the XML mediator as an interface between workflow engines and activities. This way XML technology allows clients to participate in Internet workflows and e-services without any special setup and with user-acceptable performance.

Future work includes the practical proof of concept that the mediator can also be leveraged for the communication between different WFMSs (API 4 of the WfMC reference architecture). Furthermore, we plan to extend the mediator to support transactional communication (i.e., using 2PC among workflow engines, applications, and business-object servers). So, in the long term, the mediator should evolve into a standard middle tier for data exchange between WFMSs and applications as well as between different WFMSs.

References

[ACM90] ACM Computing Surveys, Special Issue on Heterogeneous Databases, Volume 22, Number 3, 1990

[AFH+99] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, N. Weiler: WISE: Business to Business E-Commerce, Int'l. Workshop on Research Issues in Data Engineering (RIDE), Sydney, Australia, 1999

[BCL+00] C. Bornhövd, M. Cilia, C. Liebig, A. Buchmann: An Infrastructure for Meta-Auctions, Int'l. Workshop on Advanced issues of E-Commerce and Web-Based Information Systems (WECWIS), San Jose, California, 2000

[CHD+99] Q. Chen, M. Hsu, U. Dayal, M. Griss: Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation, Technical Report, Hewlett Packard Software Technology Laboratory, 1999

[CIJ+00] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, M.-C. Shan: eFlow: a Platform for Developing and Managing Composite e-Services, Technical Report, Hewlett Packard Software Technology Laboratory, 2000, http://www.hpl.hp.com/techreports/2000/HPL-2000-36.html

[CZB+99] P. Chrysanthis, T. Znati, S. Bunjeree, S.-K. Chang: Establishing Virtual Enterprises by means of Mobile Agents, Intl'l. Workshop on Rea´search Issues in Data Engineering (RIDE), Sydney, Australia, 1999

[DKO+98] A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth (Eds.): Workflow Management Systems and Interoperability, NATO Advanced Study Institute, Springer-Verlag, 1998

[Fra99] P. Fraternali: Tools and Approaches for Developing Data-Intensive Web Applications: A Survey, ACM Computing Surveys, Vol. 31, No. 3, 1999

[GHS95] D. Georgakopoulos, M. Hornick, A. Sheth: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, Vol. 3, No. 2, 1995

[GMW+99] M. Gillmann, P. Muth, G. Weikum, J. Weissenfels: Benchmarking of Workflow Management Systems (in German), German Conf. on Database Systems in Office, Engineering, and Scientific Applications (BTW), Freiburg, Germany, 1999

[GSC+99] D. Georgakopoulos, H. Schuster, A. Cichocki, D. Baker: Managing Process and Service Fusion in Virtual Enterprises, Information Systems, Vol. 24, No. 6, 1999

[GWS+00] M. Gillmann, J. Weissenfels, G. Shegalov, W. Wonner, G. Weikum: A Goal-driven Auto-Configuration Tool for the Distributed Workflow Management System Mentor-lite (Demo Description), ACM SIGMOD Conf. on Modeling of Data (SIGMOD), Dallas, Texas, 2000

[GWW+00] M. Gillmann, J. Weissenfels, G. Weikum, A. Kraiss: Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems, Int'l Conf. on Extending Database Technology (EDBT), Konstanz, Germany, 2000

[Har87] D. Harel, State Charts: A Visual Formalism for Complex Systems, Science of Computer Programming, Vol. 8, 1987

[HG97] D. Harel, E. Gery: Executable Object Modeling with Statecharts, IEEE Computer, Vol. 30, No. 7, 1997

[HLG+00] Y. Hoffner, H. Ludwig, C. Gülcü, P. Grefen, Architecture for Cross-Organisational Business Processes, Int'l. Workshop on Advanced issues of E-Commerce and Web-Based Information Systems (WECWIS), San Jose, California, 2000

[IONA] IONA Technologies, http://www.iona.com

[JB96] S. Jablonski, C. Bussler: Workflow-Management, Modeling Concepts, Architecture and Implementation, International Thomson Computer Press, 1996

[KSD99] G. Kaiser, A. Stone, S. Dossick: A Mobile Agent Approach to Lightweught Process Workflow, Int'l. Process Technology Workshop (IPTW), Villars de Lans, France, 1999

[LBS+99] H. Ludwig, C. Bussler, M.-C. Shan, P. Grefen: Cross-Organizational Workflow Management and Co-ordination, WACC'99 Workshop Report, http://www.zurich.ibm.com/%7Ehlu/WACCworkshop/Summary.html

[Ley95] F. Leymann: Workflows Make Objects Really Useful, Int'l. Workshop on High Performance Transaction Systems (HPTS), 1995

[LR99] F. Leymann, D. Roller, Production Workflow: Concepts and Techniques, Prentice Hall, 1999

[Moh99] C. Mohan, Workflow Management in the Internet Age, Tutorial, Workshop on Next Generation Information Technologies and Systems (NGITS), Zikhron-Yaakov, Israel, 1999, http://www-rodin.inria.fr/~mohan

[MWG+99] P. Muth, J. Weissenfels, M. Gillmann, G. Weikum: Integrating Light-Weight Workflow Management Systems within Existing Business Environments, Int'l Conf. on Data Engineering (ICDE), Sydney, Australia, 1999

[MWW+98a] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, Enterprise-wide Workflow Management based on State and Activity Charts, in [DKO+98]

[MWW+98b] P. Muth, D. Wodtke, J. Weissenfels, A. Kotz Dittrich, G. Weikum: From Centralized Workflow Specification to Distributed Workflow Execution, Intelligent Information Systems, Special Issue on Workflow Management, Vol. 10, No. 2, 1998

[Pap99] M. Papazoglou: The Role of Agent Technology in Business to Business Electronic Commerce, in: M. Klusch, O. Shehory, G. Weiß (eds.): Cooperative Information Agents III, Lecture Notes in Computer Science (LNCS), Vol. 1652, Springer, 1999

[SDD+97] M.-C. Shan, J. Davis, W. Du, Y. Huang: HP Workflow Research: Past, Present, and Future, Technical Report, Hewlett Packard Software Technology Laboratory, 1997

[SOAP] Microsoft Developer Network (MSDN): Simple Object Access Protocol, http://msdn.microsoft.com/workshop/xml/general/soapspec.asp

[TPC] Transaction Processing Performance Council, http://www.tpc.org

[Vei99] J. Veijalainen: Transactions in Mobile Electronic Commerce, in: G. Saake, K. Schwarz, C. Türker (eds.): Transactions and Database Dynamics, Lecture Notes in Computer Science (LNCS), Vol. 1773, Springer, 1999

[WfMC] Workflow Management Coalition (WfMC), Reference Model and Glossary, http://www.wfmc.org

[WfMC00] Workflow Management Coalition (WfMC), Workflow Standard - Interoperability Wf-XML Binding, Document Draft, January 2000, http://www.wfmc.org

[WfMC98] Workflow Management Coalition (WfMC), Workflow and Internet: Catalysts for Radical Change, White Paper, June 1998, http://www.wfmc.org

[WGR+00] J. Weissenfels, M. Gillmann, O. Roth, G. Shegalov, W. Wonner: The Mentor-lite Prototype: A Light-Weight Workflow Management System (Demo Description), Int'l. Conf. on Data Engineering (ICDE), San Diego, California, 2000

[WW97] D. Wodtke, G. Weikum, A Formal Foundation For Distributed Workflow Execution Based on State Charts, Int'l Conf. on Database Theory (ICDT), Delphi, Greece, 1997

[XML] Microsoft Developer Network (MSDN): XML Developer Center, http://msdn.microsoft.com/xml/default.asp

[XSQL] Oracle Technology Network, http://technet.oracle.com/tech/xml/xsql_servlet