XML repository and Active Views Demonstration

J.C. Mamou, C. Souza

S. Abiteboul, V. Aguilera,
S. Ailleret, B. Amann,
S. Cluet, B. Hills,
F. Hubert, A. Marian,
L. Mignet, B. Tessier.
A.M. Vercoustre

T. Milo

ArdentSoftware
Denver, Colorado

INRIA/Verso
Rocquencourt, France

Computer Science Dept.
Univ. of Tel. Aviv

1 Overview

The goal of this demonstration is to present the main features of (i) Axielle, an XML repository developed by Ardent Software [3] on top of the O_2 object-oriented DBMS and (ii) the ActiveView system which has been built by the Verso project at INRIA [1] on top of Axielle.

The demonstration is based on a simple electronic commerce application which will be described in Section 2. Electronic commerce is emerging as a major Web-supported application. It involves handling and exchange of data (e.g. product catalogs, yellow pages, etc.) and must provide (i) database functionalities (query language, transactions, concurrency control, distribution and recovery) for the efficient management of large data volumes and hundreds of users as well as (ii) standard data storage and exchange formats (e.g. XML, SGML) for the easy integration of existing software and data.

The ActiveView system combined with the Axielle XML repository enables a fast deployment of electronic commerce applications based on a new high-level *declarative* specification language (AVL), advanced database technology (object-oriented data model, XML query language, notifications), Web standards (HTTP, HTML) and other Internet compliant technologies (Java, RMI).

The prime motivations for building on the emerging XML technology are that XML will be a standard for exchanging semi-/structured information over the Internet and will be supported by many software components in form of compilers, import/export filters, query interfaces and sophisticated editors and browsers. Besides the usage of an XML repository, the ActiveView system integrates a number of other modern database technology:

XML Views: Typically, in electronic commerce application there exists different kinds of users who access data with different points of view depending on their access rights and on the nature of the activity they are presently involved in. These aspects of ActiveViews are built on our experience with O_2-Views [5], a system we developed at INRIA. Roughly speaking, we modified O_2Views by (i) considering XML data and query languages and (ii) adding sophisticated access and update facilities.

Active databases: An active database is a database that can respond to certain *events* according to certain *triggers*. Very sophisticated active mechanisms have been proposed with modest success in industry. (Some database systems provide limited triggering mechanisms.) Electronic commerce applications are, by nature, very active, e.g., when a new order is received a number of actions may have to be started. We introduce active features in a somewhat minimal way based on a notification mechanism and a rule manager.

Although we primarily target electronic commerce applications, our approach can obviously be applied to a wide range of other applications involving (i) sharing of data and (ii) cooperation of a number of actors connected via a network.

2 Demonstration

2.1 A Simple E-Commerce Application

The demonstrated electronic commerce application involves several types of *actors*, i.e. a customer, a vendor, a dispatcher. It also manages a significant amount of *data* :

- a products catalog provided by CAMIF, a French company specialized in mail-order business,
- a list of promotions information (typically viewed by customers and updated by vendors),
- the list of current customers and available vendors (used by the dispatcher).

Each of the actors *view* different parts of the data (e.g. each of the customers can see only their own orders and the promotions relevant to their class, while vendors view all the orders and promotions), each may perform different *actions* on the data, and have different *access rights* (e.g. promotions can be updated only by certain vendors). Also, the *'freshness'* of data may differ, e.g. when promotions are updated, the customers screen is immediately updated by the new data, while refresh of catalog portions viewed by customers are deferred until explicit requests.

The activity of each of the actors consists of several *sub-tasks*, e.g. a customer can *search* the catalog, *order* some selected item, *change* an existing order. Observe that each of these sub-tasks requires only part of the data and actions available for the given actor, and that the same piece of data can be used in several not necessarily consecutive sub-tasks, e.g. the *search* task queries catalog while the *order* and *change order* activities use the orders list and add, or update, resp., elements to the list.

Finally, actions performed by an actor in a certain task may initiate other tasks of the same actor (e.g. a 'perform search' action in the customer's *search* sub-task may query the catalog and then move the customer to the *order* sub-task where he can view the selected items and order them), or effect other actors (e.g. when a vendor updates a promotion, the screen of the relevant customers is refreshed). The system may also want to log some of the actors operations, providing a *trace* for later analysis.

2.2 Demonstration Architecture

The ActiveView system uses Ardent Software's XML repository that provides all the usual database features such as persistency, versioning, concurrency control, etc. All data stored, exchanged and viewed by users are XML data which are accessed by a standard DOM [6] API. The system is based on a client/server architecture. An Active View application consists of several independent clients communicating among themselves and with the repository server through notifications. Clients are programmed in Java and communicate with the server using the DOM interface.

Figure 1 shows the various components of the demonstrated application (obviously, several such applications may run simultaneously on the same server).

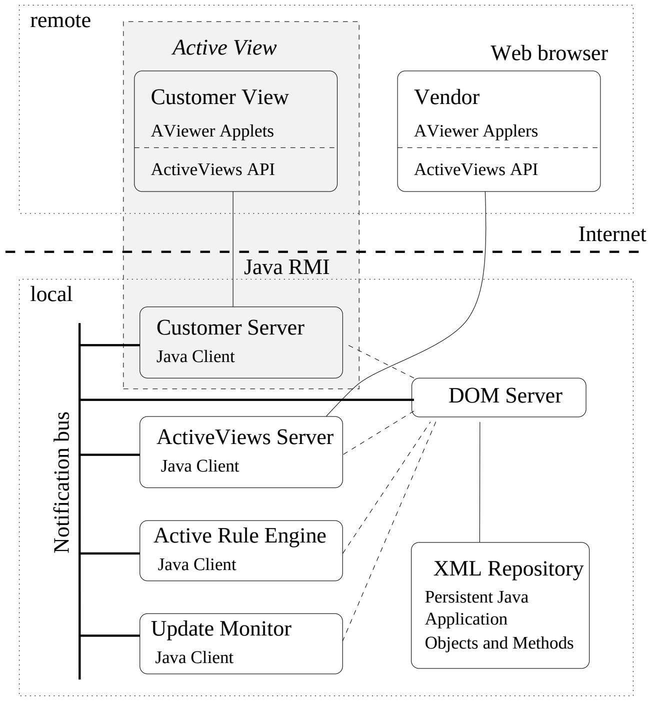

Figure 1: Demonstration Architecture

The active view application manager consists of a set of modules managing: (i) connection and authentication, (ii) tracing, (iii) update propagation and (iv) active rules. More precisely:

1. The connection/authentication module is in charge of authenticating users and giving them the means to create[1] or quit a view (via the network). Whereas we currently use a simple internal authentication mechanism, the integration of external authentication services is straightforward and outside the scope of the demonstration.

2. The update monitor propagates update events generated by the XML repository. Essentially, it transforms primitive update events (modification of a DOM node) into higher-level events concerning the modification of query results (views) and XML documents used by other components.

3. The tracing module keeps a log of specified events. These events are generated by the application or by some views.

4. The active rule module manages a programmer-specified set of rules. These rules are fired according to events and may have impact on the repository and on some or all of the active views.

[1] An active view is started from the Web using a particular URL. The system provides a form asking for the name of the system, the name of the view application, and the type of the view.

They form the essential components to specify a business model.

The last two modules rely heavily on a stream of notifications managed by the repository server that enables the interaction between views at run time. These notifications are generated according to the views specification (Section 2.3). Two kinds of events can be notified: (i) events generated by the repository server after the creation/deletion/update of objects and (ii) user defined events generated by the clients. The support for this is provided by the O_2 notifications mechanism.

An active view is basically an object of our application. In the current version of the system, it is implemented in Java. The object belongs to a (subclass of a) particular class called *ActiveView*, which is an abstraction of the class we use in the actual implementation. This class contains certain instance variables, including in particular the *owner* instance variable that is used for storing information on the user initiating the view. The class also has some methods such as transaction/commit/abort to handle a transaction mode, or init/quit/sleep/resume. An active view is related to an actual Web browser opened by a user of the system. Some views independent of any interface may also be introduced, e.g., for bookkeeping. An active view has access to the repository as well as to some local data (the instance variables of the view object). It reacts to user commands and may be refreshed according to notifications sent by the server or the view manager. The methods available on a view depend on the users access rights and may allow him/her to read, load, write, etc. part or the whole the data it sees.

The demonstrated user interfaces are implemented as HTML documents with embedded Java applets (see Figure 2). Our goal is to switch to XML as soon as XML browser supports the needed dynamic features. By default, for each activity and user there exists one HTML page. The embedded applets are built on top of an API generated by the system according to the view specification. Although the system generates default interfaces, the application programmer may redefine/customize them using the generated API that captures the semantics of the application.

The repository server, the active views and the corresponding interfaces run on different machines. Typically, the interface corresponds actually to some Web browser on any PC connected to the Internet. The view data is obtained by check-in/check-out, so repository changes are not in general immediately propagated to the view although they can be propagated, if specified by the programmer. On the other hand, the view and the interface see the same data.

2.3 Specification Language and Application Generator

The above application can be specified in the high-level ActiveView Language (AVL) [1] we have defined for describing electronic commerce applications. Acting as an application generator, the Active View system can then generate an actual application that captures the semantics of the above specification and allows the different users to work interactively on the specified data for performing the given set of controlled activities.

AVL specifications are given to a compiler that uses information stored in the repository to generate: (i) an Active View manager (unique per application), (ii) a set of active views, and (iii) users interfaces. An Active View specification is a declarative description of applications of the above nature. It specifies, for each of the actor types participating in the application,

1. the data and operations available to this particular actor and these with a sophisticate access control. The specification of the view is based on AVQL, a query language for XML we have implemented in the spirit of Lorel [2] and XML-QL [4].
2. the kinds of activities this actor may be engaged in and the subset of data and operations available in each;
3. some active rules that notably specify the sequencing of activities (a *workflow* component) but also the events this actor wants to be notified of (a *subscription* component) and those that have to be logged (a *tracing* component).

References

[1] S. Abiteboul, B. Amann, S. Cluet, A. Eyal, L. Mignet, and T. Milo. Active views for electronic commerce. In *Int. Conf. on Very Large DataBases (VLDB)*, Edinburgh, Scotland, September 1999.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1), April 1997.

[3] Ardent Software. *http://www.ardentsoftware.com.*

[4] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Xml-ql: A query language for xml. http://www.w3.org/TR/NOTE-xml-ql/.

[5] C. Souza, S. Abiteboul, and C. Delobel. Virtual schemas and bases. In *Proc. EDBT, Cambridge*, 1994.

[6] W3C. Document Object Model (DOM). http://-www.w3.org/DOM.

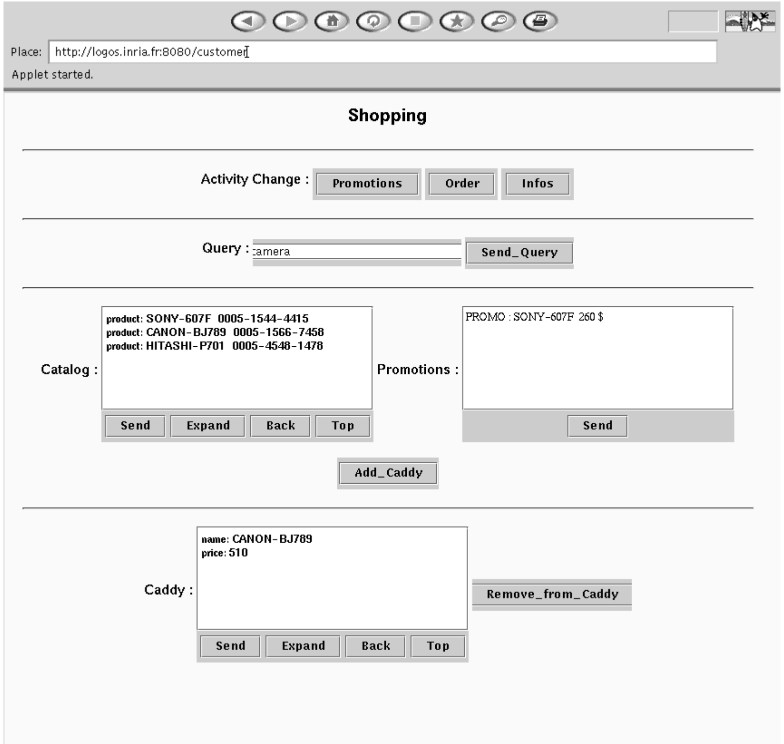

Figure 2: Example: The client user interface