webXice : an Infrastructure for Information Commerce on the WWW

Andreas Wombacher, Paraskevi Kostaki, Karl Aberer
GMD-IPSI
German National Research Center for Information Technology
Integrated Publication and Information Systems Institute
Dolivostrae 15, D-64293 Darmstadt
wombach, kostaki, aberer@darmstadt.gmd.de

Abstract

Systems for information commerce on the WWW have to support flexible business models if they should be able to cover a wide range of requirements imposed by the different types of information businesses. This leads to non-trivial functional and security requirements both on the provider and consumer side, for which we introduce an architecture and a system implementation, webXice. We focus on the question, how participants with minimal technological requisites, i.e. solely standard Web browsers available, can be technologically enabled to participate in the information commerce at a system level, while not sacrificing the functionality and security required by an autonomous participant in an information commerce scenario. In particular, we propose an implementation strategy to efficiently support persistent message logging for light-weight clients, that enables clients to collect and manage non-reputiable messages as proofs. We believe that the capability to support minimal system platforms is a necessary precondition for the wide-spread use of any information commerce infrastructure.

1. Introduction

Electronic systems are becoming more and more significant in industrial, commercial and scientific applications. As modern markets move rapidly onto electronic platforms, E-Commerce and eBusiness are becoming key terms in today's economy reflecting this tendency. E-Commerce addresses the trading of physical goods, such as books, food, computers and appliances. Trading information goods, like news, software, or reports, is even more attractive over electronic channels, since goods can be distributed through the same infrastructure. We will use the notion of *information commerce* for this specialized form of electronic commerce.

Models and infrastructure for Information Commerce, constitute an area of active research and development that comprises many technical challenges. These are related to delivery, copyright issues, tamper-resistance, adequate payment facilities, non-repudiation and proof of actuality of the information. A limitation of the current situation is that users are overwhelmed by an enormous quantity of unstructured, uncertified data. We argue that this may be acceptable for the average citizens, it is certainly not for professional users. Organizations, professionals but also private citizens willing to gain a profit from the knowledge of information or to rely important decisions on information demand value-added services like personalization, evaluation, categorization and combination of information. In addition, they require that the origin of the information is trusted and that the information is fresh, in order to ensure that they base their decisions on up-to-date and accurate information.

Within the European research project OPELIX [23] we develop an infrastructure for information vendors who sell original of information, information mediators, who buy, recombine and resell information, and information brokers who provide directories of information vendors together with added-value information. We assume that information has an associated value, that requires controlled access in an individualized manner and guarantees concerning the quality and authenticity of the information, but that the information is not that sensitive, that it requires strong protection mechanisms to disable unauthorized access. The different properties associated with an information product and the corresponding interaction between buyers and sellers will be specified in a highly configurable business offer specification language. This is an adequate assumption for many application types, like portal sites, electronic new services, stock market information services, software evaluation services, or directory services.

In this paper we describe webXice [24], a first implementation of a communication and information distribution platform that supports the requirements described.

Within the webXice engine (web-enabled extended ICE protocol) the XML-based Information and Content Exchange (ICE) protocol, a W3C note, is implemented and

extended. The ICE protocol [10] describes how information providers and consumers communicate to exchange information using a differentiated information exchange model. ICE has been proposed by various vendors of systems supporting information business, mainly for the publishing industry and for catalog data exchange.

The webXice infrastructure builds on existing Web technology. We will specifically describe which client-side architecture has been developed in order to enable light-weight clients to participate in the communication required to perform information commerce using an interaction protocol like ICE. Whereas the server implementation can rely on full fledged server technology, including database services and server-side components, a client solely equipped with a standard Web browser has to deal with a number of limitations like

- No capability to actively respond on requests
- Persistency mechanism are lacking (e.g. database systems)
- The runtime environment has limited capabilities (e.g. Web browser, Java)
- Restricted access the operating system resources (e.g. through applets)

Despite of those limitations we will describe how a minimal support with regard to supporting the business logic and specifically security can be provided. The minimal client is enabled to correctly participate in the message exchange, to verify correctness of messages and to log messages persistently to collect proofs in case of dispute with the business partners.

The contents of the paper are organized as follows. In Section 2 we give a high-level description of an information commerce marketplace, describing the overall scenario we have in mind. Then we describe in Section 3 the webXice architecture in two typical instantiations covering the scenario's requirements. Section 4 describes the main models realized within the webXice system, including the message exchange model, the business process model and the logical inter-message relationships. The description of the implementation of the various webXice components is given in Section 5. Section 6 describes in detail what solutions have been chosen to instantiate webXice on a light-weight client. In Section 7 we illustrate the use of the system by means of a sample scenario. Finally, we conclude with related work in Section 8 and an outlook on future work in Section 9.

2. High-level description of an electronic information commerce marketplace

In Figure 1 we illustrate the main functions of an infrastructure supporting an electronic marketplace for information commerce performing business in an open environment with several parties involved in processing several trades. In this scenario an information mediator obtains information from two suppliers and then sells the enriched information to a user. Trusted third parties are involved in the trading process in order to enable the issueing of certificates and the financial clearing. Timely delivery of information is critical (indicated by the clock symbol) as freshness of information is an important quality measure.

Before an actual delivery of information can be performed a search and negotiation phase is performed, which is indicated by the functions in the box. This phase, in turn, consists of:

- *searching* business offers that suit the users' demands;
- *matching* the information offer and the user' demands;
- *negotiating* the open parameters in the business offer.

Important aspects of the negotiation besides payment and delivery terms, are the copyright terms implied by the information offering, quality parameters and the means used to protect and authenticate the information good. After an offer has been negotiated, the two parties settle a business contract, which is then executed. The business offers and business contracts are expressed in a "business offer description language". The business offer description language provides models to define which information services are, respectively, offered by the business (business offers) and requested by the users. A business offer describes the type of information sold by a business and under what conditions, with which quality guarantees and with which subscription facilities the information is made available.

The actual execution of business contracts requires a number of functions that perform the following operations by interpreting the "business offer description language":

- *Delivery* of the information according to the contract terms to the user, both in push and pull mode, within time constraints and with the desired information quality.
- *Payment*, by involving a financial clearing organisation.
- *Authentication*, by involving an authentication service.
- *Mediation*, for processing business offers that provide information combined from different sources and with added-value.

A simple sample scenario for such a marketplace, that we have implemented with the approach described in this paper, is electronic procurement, where catalogue information is exchanged between different parties. Typically product specific data is stored centrally and maintained within a legacy system of the manufacturer, who can provide his

customers, e.g. vendors or stores, with the updated product information electronically. Large companies can afford a continuous connection to the Internet and typically maintain servers available from the Internet (e.g. for hosting the web presence), something that small companies not necessarily have. Furthermore, large companies invest more in system integration than small ones, because they gain a higher benefit from automated and error-free integration of high volume data. To support both types of companies the manufacturer should electronically provide the updated product information in an appropriate way. The resulting solution has to support communication between servers as well as communication between a server and a minimal client. The provision of this information has to be performed according

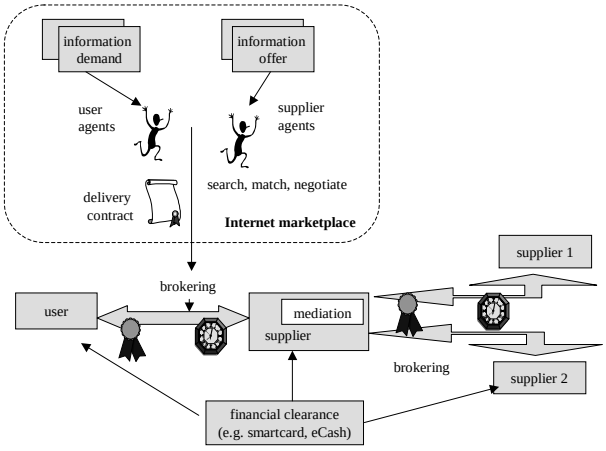

Figure 1. Main functions of an electronic market place for information commerce.

3. Architecture of an information commerce system

In this section we outline the architecture of webXice, a prototype system for implementing the scenario described in the previous section. We based this system initially on an existing business offer specification language, namely the ICE (Information and Content Exchange) protocol. The ICE protocol describes a communication protocol for use by content providers and their users and will be described in detail later. Though it is limited in scope with respect to our purposes, e.g. no payments are supported, it is sufficient as a starting point and is currently being extended with respect to additional requirements.

The webXice architecture is designed in a way that allows to make maximal reuse of existing infrastructures for content representation, information dissemination, payment, copyright, authentication, and security. Only the specific components required to process business offer descriptions are developed.

The system is based on a distributed architecture. In particular, it is composed of a number of servers at the information providers side and a number of clients, one for each user. The core of the system is the webXice library that provides the basic functionality that is common to both the client and server side of the system. The detailed functions of the webXice library will be described in Section 4. The core library contains a business offer language interpreter, which interprets a business offer specification for a certain information product and executes the corresponding workflow. In doing that it uses different services for delivery, payment, authentication and copyright that are accessed through defined API's. By this architecture the system is as open as possible for extensions and as modular as possible for the specific configurations of the different users. The services used by the webXice system are the following.

- Services for information distribution. Though contents can in principle be delivered directly within messages exchanged by the webXice systems, this is not necessarily the most efficient way of delivery. As in the ICE protocol we also allow to send references to content, that are used by the consumer to access the delivered information. In addition, specialised delivery systems, like push systems [11,12] can be integrated, in order to optimize the physical delivery of large contents over the Internet.

- Payment services. Minipay [2,3] is one approach to standard micro-payment protocol we use for webXice. It is originally designed to support electronic payment on the Internet. However, this model can be adapted to other payment schemes, like pay-per-view, volume-based, or flat fee schemes. For larger payments, also credit card based payments, like SET [13], can be integrated.

- A security infrastructure. The security infrastructure is based on the standard X.509v3 public key certificates. It is used for signing of messages, according to DigSig [9], and the signing of applets in order to provide secured applets.

Based on the webXice library different instantiations of webXice systems are realized supporting the different roles in the information commerce scenario, as well as taking into account different system platforms. In the simplest configuration, a merchant server interacts with a light-weight client. This configuration would typically be used in a B2C application. This configuration is illustrated in Figure 2.

The merchant side of this configuration consists of a supplier system that has access to a payment system and a

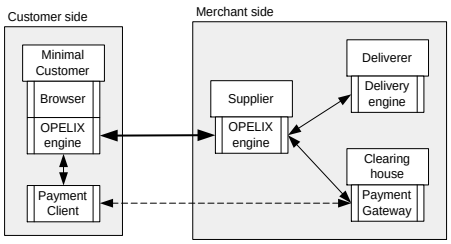

Figure 2. A light-weight client architecture

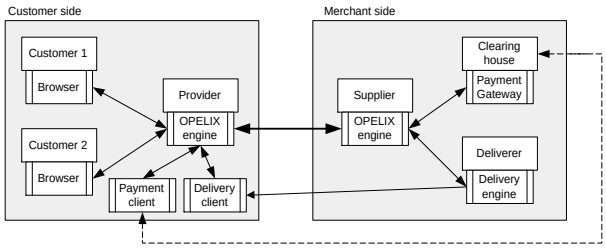

Figure 3. A B2B architecture

delivery system. The payment and delivery systems can be owned by the merchant or be outsourced to service providers. The delivery system will in this configuration be a Web server that can be accessed by the customers browser. The payment system performs the clearing of payments, and provides accounting facilities. Besides the basic processing capabilities for the interaction with a customer provided in the current version of webXice, the supplier system will be enabled to offer information products and mediate information products, i.e. combine information from other merchants to create new information products.

Technically, the current prototype of the merchant side is implemented using a servlet infrastructure. The system is running without problems on different platforms, like the Apache web server and the jserv servlet engine, the Netscape Enterprise web server with the jrun servlet engine and the IBM WebSphere application server.

The client side of this configuration is implemented as a ”thin” client, that requires as only infrastructure a Web browser and a payment client, that provides access to the payment gateway of the server. In addition to the core functions of webXice this system provides facilities for interactive access, through a simple web interface via the web browser. The merchant side has to take measures to communicate with such a client, also in situations where it is partially disconnected.

How the functionality of a webXice client can be realized with this minimal infrastructure without sacrifycing functionality and security will be discussed in greater detail in Section 5.

For customers that represent larger organizations with corresponding infrastructure, e.g. in B2B applications, a different architecture is chosen, that is illustrated in Figure 3.

Here on the customer side webXice runs on an Intranet server that can be accessed by the different members of the organization to perform the transactions. The Intranet server in turn is connected through a stable connection to the suppliers server. Also the Intranet server of the company keeps all the necessary information and logs all the transactions that are taking place. Since in this scenario the customer side may afford to install specialized delivery systems, also more advanced methods of content delivery can be supported, like push systems or dedicated ICE engines of external providers.

4. The webXice library

The webXice library provides the core function of the information commerce infrastructure. It implements the information trading process through message exchanges among the trading partners and calls to the backend systems for payment, delivery and authentication. For communication it uses the Internet as infrastructure. For message representation it uses the XML document format. The webXice engine realizes the following main processing phases as illustrated in Figure 4.

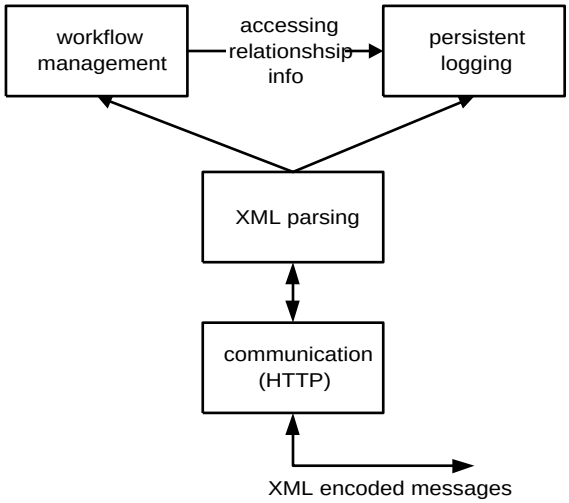

Figure 4. Model of the internal workflow

1. Message exchange: the webXice engine provides generic capabilities to exchange messages according to a generic request/response protocol over the Internet communication infrastructure (HTTP) and to convert messages back and forth from the document representation (in XML) to an internal object-oriented representation (in DOM [27]) for further processing.

2. Business process: the webXice engine implements a business process for trading information. Each step in

this process consists of a message exchange and an interaction with backend systems or the end user. The engine controls whether incoming messages are in accordance with the current state of the business process and initiates the sending of the correct messages in the current state, such that the participant behaves correctly according to the business protocol.

3. Persistent logging: the webXice engine keeps a persistent representation of the business process state by storing the exchanged messages. In addition it provides management functionality to efficiently access the persistent state for analysis and verification purposes.

In the following we describe in more detail each of the processing phases and their underlying models.

4.1. Business process model (ICE)

The business process supported by the webXice engine depends on the business model used for information trading. Thus a flexible language for describing such processes is required, a so called business offer description language. In the current version of webXice the main focus of webXice was on the elementary infrastructure required by an information commerce system rather than on developing sophisticated languages for describing business processes. Thus we have used an established standard information commerce language for the implementation of webXice, namely the ICE standard.

The ICE (Information and Content Exchange) protocol [10] serves us two purposes. First, we use and extend the basic communication patterns of the ICE standard as a basis for the model of controlled message exchanges used by webXice, which will be described in Section 4.2. Second, we will use the specific ICE message types as one possible instantiation of a business offer language. Since ICE has been designed for closed marketplaces, this language will be extended in future versions in order to support additional functions, like payment and authentication. The basic communication patterns will however remain, also for the extended languages, and thus the subsequently described webXice infrastructure can be reused. We describe in the following the main characteristics of this protocol shortly.

The ICE protocol is a communication protocol for use by content providers (called syndicators) and their users (called subscribers). The ICE protocol is a closed user group protocol, which defines the roles and responsibilities of the participants, the syndicators and subscribers, the message formats and the method of content exchange, but leaves other dimensions like payment, authentication or metadata for content description open as orthogonal dimensions. The two main phases of the protocol are subscription establishment and management and data delivery.

An interaction between syndicator and subscriber starts by establishing a subscription. Typically the subscriber first obtains a catalog of offers from the syndicator and then subscribes to a particular offer by proposing it to the syndicator. Alternatively the two parties may engage in a parameter negotiation protocol. After the subscriber subscribes to a particular offer the data delivery phase starts. ICE uses a package concept. Packages encapsulate contents of arbitrary type. A sequenced package model allows syndicators to support both incremental and full update models. ICE also defines push and pull data transfer models and detailed temporal and quantitative constraints on delivery.

One extension of ICE that is currently under development is the integration of payment messages. Such an extended language then allows the provider to flexibly describe a variety of different payment models, including the well known ones like detail-based, pay-per-view, volume-based, quality-dependent, time-based, and flat subscription fees, or a combination of these.

4.2. The webXice communication model

The ICE protocol is a request/response protocol that allows for fully symmetric implementations, where both the syndicator and subscriber can initiate requests. This request/response model of ICE is mapped to the HTTP post/response model. WebXice supports the symmetric ICE request/response model generically by abstracting from the specific message types of the ICE standard. In addition it supports and generalizes the unsolicited message model introduced in ICE, which we describe briefly in the following.

Depending on the available infrastructure a server may not be able to issue requests to clients. This is for example the case for a minimal client (also called a minimal subscriber) implementation based on Web browsers only. A way to address this problem is proposed in the ICE protocol specification by means of the unsolicited message concept. A server can set an unsolicited flag within a response. So it indicates that it wants to send a request to the client. In response, the client sends an unsolicited-now message, which is then answered by the unsolicited-request of the server (this is a real response with regard to the underlying infrastructure). This response is then answered by the client sending an unsolicited-response (this is a real request in the underlying infrastructure). The server can now continue sending unsolicited-request or it can stop the unsolicited communication by sending an unsolicited-finish message to the client. We depict this mechanism formally by an automaton describing the possible message sequences in Figure 5. The arrows marked by the boxes belong to the requests of the underlying infrastructure, while the others are responses.

For general applicability in webXice repetitive sequences of unsolicited-requests and unsolicited-responses are possible. Also a unsolicited-finish response message is

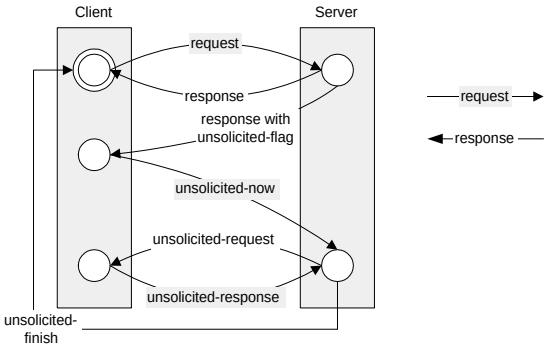

Figure 5. The webXice communication model

introduced to remain consistent with strict request-response model. (This corrects in fact a minor flaw in the current ICE specification.)

4.3. Logging of messages

All messages that are exchanged are persistently stored in a log. For example, when the messages are digitally signed (e.g. according to the DigSig [9] standard), they can be later used as non-repudiable evidence in case of disputes, if the business protocol is designed carefully. An analysis of what properties of a protocol are sufficient in order to enable non-reputiability in an open market environment is given, for example, in BAKO [4,5].

In order to analyze message exchanges, to verify their correctness or to provide proofs in case of disputes, the persistent message log has not only to record the messages exchanged, but also to be able to manage the relationships that exist among the messages. There exist different types of relationships, that need to be considered:

1. Communication relationships: these are the request/response pairs and the sequences of messages exchanged in an unsolicited request processing. They are based on the generic communication model that webXice is based on.

2. Semantic relationships: they are based on the references used in messages to relate messages to earlier messages for business purposes. In the ICE protocol as used in webXice the following references occur: (a) offer messages in the offer phase may be related to specific catalogs, (b) messages of the subscription phase refer to a specific subscription-id (example messages are ice-change-subscription, ice-cancellation, ice-offer with subscription-id), (c) messages in the delivery phase are related through the package sequence model of the ICE protocol as well as by the subscription-id.

Based on these relationships traces of consecutive messages referring to each other need to be produced by a log manager upon request.

5. Implementation of a light-weight webXice client

The implementation of a server side webXice engine can rely on a powerful infrastructure, including database systems and application servers. Such an approach is neither easy to use for inexperienced users, nor flexible and several problems appear whenever there is an upgrade. Since we are interested to keep the information commerce market infrastructure as open as as possible, we should, especially on the client side, not be forced to rely on such an infrastructure. Therefore we explore the architecture of a light-weight webXice engine, that can be fully implemented based on a typical Internet client environment consisting of a standard Web browser, without sacrificing the needed functionality that has been described in Section 4 and without compromising security.

5.1. Implementation platform

Four software solutions were considered for the development of the proposed light-weight webXice client: (i) applets, (ii) plug-ins, (iii) activeX controls and (iv) Java applications running on the client side (probably accessing a local database via JDBC). Java applications provide full access to system resources like file system and local databases, but may be operating system dependent, and require the customer to install proprietary software and to manually start up application programs. ActiveX is proprietary to the Microsoft platform and thus limited in use. PlugIns are not necessarily platform dependent and have limited access to local resources. Applets can be flexibly downloaded, thus the user is relieved from version management, they require no local resources, and with the Java security model (sandbox), access to local resources can be controlled. The main limitation we have to consider for applets, is keeping them small in order to avoid long download times. Thus, it turned out that applets are the most appropriate choice for our purposes.

5.2. Storage management

The next step is to examine the different types of persistent storage accessible by applets. The main options we have at hand here are:

- local database system
- local file system
- cookies, which can be addressed directly or via JavaScript

A local (standard) database system provides high storage capacity and functionality, but is typically not available on a

thin client installation and imposes administrative overhead. Thus we omitted this possibility.

Allowing foreign code to access the local file system is a critical point. Therefore the Java environment provides the sandbox model. The Java sandbox is responsible for protecting a number of resources (like, net access and file access). It does so at a number of levels depending on the visitors confidence [1]. Typically a signed applet is allowed to access more resources (e.g. like the local file system) than an unsigned one.

Cookies are used to store server information on the client side. They are easily accessible through HTTP and JavaScript. Cookies can later on be retrieved, modified and deleted by all servers matching the domain name captured in the cookie. By the usage of JavaScript cookies can also be accessed by the client directly, but are then no longer readable by the corresponding server. Cookies are limited by machine and browser dependent parameters on the maximal number of stored items and the maximum size of an item.

With all this in mind, there remain three different alternatives based on applet technology for persistent message logging:

- exclusive use of cookies
- exclusive use of local file system
- use of both cookies and local file system

The first alternative is based on applets on the client side that use cookies as persistent storage. The main problem are the limitations of cookies when used to store data. First, cookies have an expiry date, which has to be set. In case only short term storage is needed, e.g. till the end of a transaction, this is not a severe restriction. Second, the number of cookies is limited, e.g., for the Netscape browser there are no more than 20 cookies allowed for a specific page and the total number of cookies stored within one file is limited to 300 cookies. Third, there exist also browser dependent limitations in the size of the cookies. For example, Netscape limits the cookie size to 4096 bytes. So cookies can be easily used without generating security problems on the client side, but the size of the persistent store is very limited and dependent on the browser type. This approach can be used for very short term business relations that require small amounts of persistently stored data for logging.

The second alternative is the use of the local file system. An applet, which is downloaded from the merchant web site accesses directly the local file system to store data persistently. The main problem here is security. As described earlier, the applet is running in a sandbox, where the user has to grant access rights to the sandbox. In case the applet has total access to the local file system it is able to read, write and delete all data. For security one can use only applets signed by a trusted third party, but this in turn may limit the function that is supported by the applets.

For practical reasons and in order to enable merchants to differentiate themselves it is desirable that merchants provide applets with merchant-specific functions. Such added-value functions could be specialized visualization of the offer messages, support of optimized subsets of the general information commerce protocol, or support for vendor-specific delivery mechanisms. Therefore we do not want to rule out the possibility that a vendor provides the specific applet that implements the basic workflow but provides additional specific functions for the optimized interaction with the customer.

To cover this we combine both approaches. We use unsigned applets from untrusted business partners, which are allowed to store their transaction data in cookies and we use standard applets from trusted third parties to make transaction data in cookies persistent via the file system. In this way we combine the ability to support flexible merchant specific applets for trading and the security that access to critical resources is granted only to trusted applets. The only drawback is that some additional coordination and communication among the two applets is needed.

5.3. Representation of the Log Data

The standard data representation mechanism we have at hand is XML respectively DOM as an object-oriented representation of XML. In addition to straightforwardly store the XML structured messages in the log, the different types of message relationships introduced in Section 4.3 need to be managed. We provide this support by representing those relationships within the data model of XML/DOM explicitely.

For representing a relationship XML/DOM provides in principle two different mechanisms, namely primary relationships, which are the containment relationships within a document hierarchy, and secondary relationships which are realized through the ID/IDREF mechanism of XML. Among these the primary relationships are more limited since they allow to represent only tree-like structures, but are more efficiently to process. Therefore we decided to exploit them as far as possible in the following way.

All the communication relationships, which are linearly ordered, are mapped to primary relationships. Since in ordinary request/response exchanges this does not impose any relationship from responses to the next request, additional semantic relationships can be mapped to the primary relationships. In the current implementation these are offer messages (ice-offer) that are sent in response to catalog messages (ice-catalog), subscription change messages (ice-change-subscription) and requests for packages (ice-get-package), which are linearly ordered in a canonical way due to the sequenced package model. The other references, like subscription-ids in delivery messages (ice-package, ice-

get-package) are implemented as secondary relationships.

Figure 6 shows the abstract representation of the primary relationships in the DOM representation derived from communication relationships, including an unsolicited message exchange, with some additional semantic relationships also including secondary relationships (dashed arrows).

Using a persistent DOM implementation, the logs can now be easily accessed for inspection using the DOM API or an XQL query interface through Java application programs provided as applets [8].

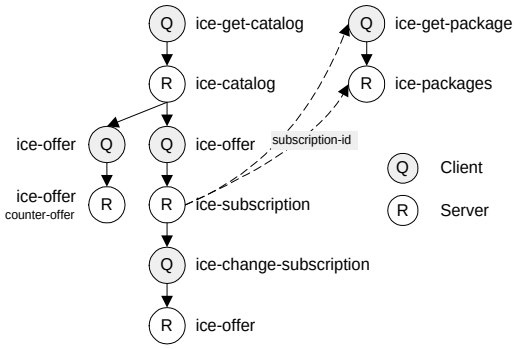

Figure 6. A complete XML message log example

6. Sample Implementation

In the following we describe an experimental implementation of a thin client. It is implemented by means of applets and uses both cookies and the local file system for storage as described in the previous section. In the following we describe the different components of the system as depicted in Figure 7.

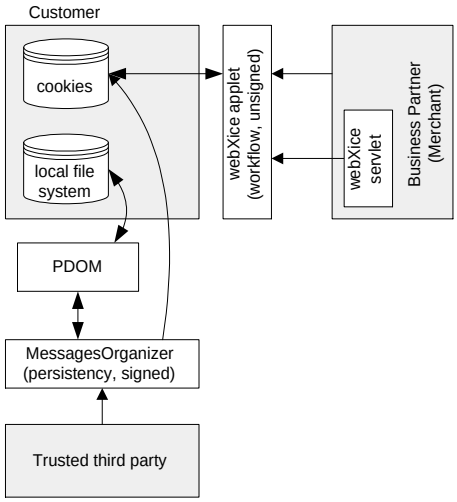

Figure 7. Architecture of the webXice client

6.1. Persistent representation of log data

As explained in Section 5.3 the messages as well as their relationships are represented in XML. The standard API for accessing XML data is DOM. In order to provide convenient access we are using a persistent DOM implementation (PDOM) [25] to store the log data, rather than storing it directly as XML document files. In this way the DOM API is also available for accessing the persistently stored log data, while the PDOM engine handles secondary storage management transparently.

The functionality of the MessageOrganizer covers the insertion of new messages, the retrieval of subscription information, and the retrieval of messages from the history.

The insertion of a new message starts with the parsing of the incoming message, and thus the creation of the DOM representation of the message and the derivation of the message relationships. The DOM representation of the relationships is created and then stored persistently together with the messages using PDOM.

The MessageOrganizer allows to retrieve an ice-subscription message related to a specific subscription-id, either through the subscription message itself or any subsequent modifications of the subscription. This function is for example needed to check the delivery conditions of a specific subscription during the delivery phase. Since the subscription-id relationship is explicitly represented in DOM it can be directly derived from this representation.

The retrieval of messages from the history supports two functions. First it allows to get all leave nodes of the current instantiation of the persistent data log. So the newest messages for each logical communication task is returned. Second we offer the possibility to get a history of the exchanged messages starting from a specific leave node of the persistent data log. This function is for example needed to provide evidence in case of a dispute.

The MessageOrganizer is also responsible to transfer information from the cookie file to the message log. Since it is able to access the local file system this function must be performed by trusted software. Therefore it is executed by a signed applet that needs to be provided by a trusted third party.

The applet is controlled in the current implementation by the user via a simple graphical user interface. The user is asked to fill in a message directory, where the respective PDOM file is stored. Furthermore the user interface provides a button, that allows to initiate the operation of copying the cookies content to the persistent data log. The applet reads each cookie of type localhost and checks whether the message is already transferred or not. If not, it is inserted into the log. The access is performed in order of entry, such that the oldest entries are read first.

For the correct working of the applet information about the type of Web Browser and the operating system is re-

quired. This information can be derived by accessing the corresponding JavaScript variables, provided by the JavaScript navigator object.

6.2. Implementation of the application logic

The workflow controller (webXice applet in Figure 9) is responsible for performing the correct message exchange according to the business process protocol and providing the user with a graphical user interfaces via HTML. Since it accesses only cookie files and the function may be specific for a specific merchant, it can be provided by the merchant as an unsigned webXice applet. The merchant can use the same controller as the webXice servlet on his side.

As an example we implemented the following ICE communication for establishing a subscription within this workflow component.

- there is a message *ice-get-catalog* send to the webXiceServlet,
- the server responds with an *ice-catalog* message,
- then the client choose an appropriate offer from the GUI and sends an *ice-offer* message,
- then the server sends an *ice-subscription* message back. (Within this example no negotiation is performed

The exchanged messages are stored in the cookie file. Therefore the applet extracts from the ICE message the elements payload ID, response/request ID, message ID and document's URL. Then the applet passes the parameters to the cookie object of the browser and places it into the cookie file. This information is used for validating whether messages are compliant the business protocol, that is checking whether the right message is sent or received. The current applet implementation uses an in-memory representation of the exchanged messages and stores the messages as persistent log into the cookie file.

Most of the implementation is completed as described, yet not fully integrated. Most importantly, the PDOM based storage component is currently used on the server side only within a servlet and will replace a currently purely file-based implementation of the MessageOrganizer applet. We expect no major difficulties with this step as applets using PDOM have been successfully used in other contexts.

7. Application Scenario

We tested the system for the sample procurement scenario described in Section 2, where catalogue information is exchanged between different parties. We focussed in this test on the information delivery components, and omitted payment functionality. The solution for the large companies feeds the delivered product information more or less directly into the internal ERP system, in the concrete case into the Net.Commerce system of IBM [15] and the Intershop system [26]. The minimal client solution allows the small vendor to access this information via an applet and to feed the product information changes manually or using scripts into his internal system. Within this scenario the product information is internally maintained in some legacy system, which we simulated by a simple database application.

8. Related Work

Systems for information commerce have been proposed for high value information goods, like DigiBox [19,20] and Cryptolopes [17,18]. In these systems the electronic goods are encrypted and stored in a container, and annotated with meta-information like description and price. After payment by the customer, the merchant provides a key for decrypting the content of the container. This concept does not require logging on the customer side beyond standard payment logging performed by the payment wallet, because the customer receives the good before paying.

For low value electronic goods, less security is required, because a potential loss of the information is less expensive. A simple shopping model is given in the "Common Markup for micropayment per-fee-links" W3C working draft [14], which describes standardized markup for the price of contents accessible through Web pages supported by multiple payment systems. This markup implements a pay-per-link business model, where the customer is charged before accessing a specific link. There are several implementations of this draft, e.g. IBM's Micro Payments [2,3].

Both types of approaches do not simultaneously address the issues arising when requiring flexible business models and trading with low value electronic goods at the same time. In case of the "Common Markup for micropayment per-fee-links" implementation based on Micro Payments also the question of persistent message logging, in order to provide some security to the consumer is not addressed.

Different E-Commerce systems, for physical goods, offer a variety of shopping models and system support. In merchant centric systems the state information is stored only on the merchant side while the customer does not have any state information or persistent logging at all. Examples of this type of systems are Intershop [7], webSphere Commerce Server [15], open market [21,22], which are mainly based on an application server while the client side is considered to be a simple web browser. The server side manages the states of the different clients and also keep track of the performed actions, not at a message level, but in an internal representation. The user can view the state information of the ongoing process, like viewing the shopping cart, inspecting the orders or tracking the delivery process [16]. The business models used within these systems are

quite static.

A proposal for a more flexible shopping model is proposed by [6]. The model assumes an external and independent shopping controller, who provides a specific business workflow to the trading partners. So the merchant can use the external shopping controller if he wants to provide his goods by the shopping controllers business workflow to gain a higher flexibility. The shopping controller holds the state information independently and triggers the different actions, like e.g. pay and deliver. While the shopping controller is an independent instance within the trade, it can store the exchanged messages persistently as an evidence for both parties.

An extension of this idea is a shopping controller on both the merchant and the client side, which can be dynamically configured. Models for having controllers on both sides are found in the Open Trading Protocol [29], used for selling physical goods, and the ICE protocol introduced in Section 4.1 for distributing electronic goods. Both protocols specify the trading workflow and the messages formats. Currently, they do not allow a flexible specification the workflows. OTP provides six baseline transactions, which are the main purchase activities, while the ICE protocol provides only flexibility in the specification of the delivery parameters. Within these models, both parties are forced to hold state information and should collect evidence. A possible implementation for the client side is proposed in this paper.

9. Outlook

Based on the infrastructure presented in this paper a number of extensions will be developed to achieve the complete functionality required by an information commerce systems. Among these are the integration of the payment functionality at a system level and the signing of the messages, such that the logged messages become non-repudiable. With the increasing popularity of mobile clients a natural question to investigate, how information commerce clients for even more light-weight platforms, like WAP [30], can be enabled. At the conceptual level our main interest is on the development of more general and flexible business offer models and specification languages that are particularly suited to represent information products, both from the content as well as from the process perspective.

References

[1] S. Oaks. *Java Security*. O'Reilly, 1998.

[2] A. Herzberg and H. Yochai. Mini-Pay - Charging per Click on the Web. In *Proc of 6th WWW Conf*, April 1997.

[3] IBM Micro Payments. http://www.ibm.com/software/ webservers/commerce/payment/mpay/index.htm.

[4] S. Andre. BAKO - Secure Internet Banking with Privacy Enhanced Mail. In *Proc. of the 7th joint European Networking Conference*, Budapest, 1996.

[5] M. Wichert. BOPD - Browse, Order, Pay and Deliver - A Purchasing Protocol on the Internet. Protocol Description, 1997.

[6] S. Ketchpel, H. Garcia-Molina, and A. Paepcke. Shopping Models: A Flexible Architecture for Information Commerce. In *Proc. of the 4th Annual Conf. on the Theory and Practice of Digital Libraries*, 1997.

[7] Intershop Enfinity White paper. http://www.intershop.com/ products/pdf/enfinity_whitepaper.pdf.

[8] GMD-IPSI XQL Engine. http://xml.darmstadt.gmd.de/xql/index.html.

[9] D. Eastlake and J. Joseph Reagle. XML-Signature Syntax and Processing. http://www.w3.org/TR/xmldsig-core/.

[10] N. Webber, C. O'Connell, B. Hunt, R. Levine, L. Popkin, and G. Larose. The Information and Content Exchange (ICE) Protocol. http://www.w3.org/TR/NOTE-ice.

[11] Minstrel. http://www.infosys.tuwien.ac.at/Minstrel/.

[12] M. Hauswirth and M. Jazayeri. A component and communication model for push systems. In *Proc. of the ESEC/FSE 99 - Joint 7th European Software Engineering Conference (ESEC) and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-7)*, September.

[13] SET. http://www.setco.org/ set_specifications.html.

[14] T. Michel et al. Common markup for micropayment per-fee-links. http://www.w3.org/TR/WD-Micropayment-Markup/.

[15] BM WebSphere Commerce Suite. http://www.ibm.com/ software/webservers/commerce/servers/index.html.

[16] UPS Package Tracking. http://www.ups.com/tracking/tracking.html.

[17] U. Kohl, J. Lotspiech, and M. A. Kaplan. Safeguarding Digital Library Contents and Users - Protecting Documents Rather Than Channels. In *D-Lib Magazine*, September.

[18] IBM Cryptolopes. http://www.ibm.com/ software/security/cryptolope/.

[19] O. Sibert, D. Bernstein, and D. Van Wie. Securing the Content, Not the Wire, for Information Commerce. with paper of research division of www.epr.com. http://www.star-lab.com/secure-the-content.html.

[20] Intertrust Home Page. http://www.intertrust.com/.

[21] Internet Commerce: The open Market Solution. Technical White Paper, Open Market Inc., November 1997.

[22] NextPage. http://www.nextpage.com/index_nojs.html.

[23] OPELIX Home Page. http://www.opelix.org.

[24] webXice. http://www.darmstadt.gmd.de/oasys/projects/webxice/.

[25] G. Huck, I. Macherius, and P. Fankhauser. PDOM: Lightweight Persistency Support for the Document Object Model. In *Proc. of the 1999 OOPSLA Workshop "Java and Databases: Persistence Options; on the 14th Annual ACM SIGPLAN Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'99)*, Denver, 1999.

[26] Intershop Home Page. http://www.intershop.com.

[27] Document Object Model (DOM) Level 1 Specification Version 1.0. http://www.w3.org/TR/REC-DOM-Level-1/.

[28] J. Clark. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt.

[29] D. Burdett. Internet Open Trading Protocol Version 1.0. Internet Draft, 1999. Commerce One.

[30] Wireless Application Protocol technical specification. http://www.wapforum.org/what/technical.htm.