# Implementation of the ebXML Registry Client for the ebXML Registry Services

Marko Topolnik, Damir Pintar, and Ivan Matasic
Department of Telecommunications
University of Zagreb, FER
Unska 3, 10000 Zagreb, Croatia
{marko.topolnik, damir.pintar, ivan.matasic}@fer.hr

*Abstract*-- **The implementation of the ebXML Registry Services is a part of a research project at the University of Zagreb, aimed at defining a strategy for the adoption of e-business in Croatia. Implementing the ebXML Registry/Repository is a vital step as it implements a standard mechanism for registering, storing and retrieving data about business partners relevant for conducting e-business. This paper describes an implementation of the ebXML Registry Client as a diagnostic tool to test and debug the ebXML Registry/Repository implementation under development. The Client can generate arbitrary submitObjects requests to the Registry Services. It guarantees the conformance of the requests with the OASIS/ebXML Registry Services Specification.**

***Index terms-- ebXML, Registry Services, Registry Client, SOAP, JAXR***

## I. INTRODUCTION

The ebXML initiative has been recognized worldwide as this decade's best chance for a globally accepted e-business platform. The effort to achieve this goal is going on in parallel at the specification and implementation levels. Early implementations serve as a source of feedback for future specification improvements.

An implementation of the OASIS ebXML Registry v2.0 is hosted at the Center for E-Commerce Infrastructure Development (CECID) in Hong Kong. Its development was started by Sun Microsystems Inc. and continued as an open-source project hosted at SourceForge. There are also many other open-source and commercial implementations.

At the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Telecommunications, there is an ongoing research and development effort, financed by Agrokor Group (the leading Croatian food production and retail company), dealing with the adoption of e-Business in Croatia. It is also a part of a larger project of the Croatian Ministry of Science and Technology, named "Networked Economy." As the initial step, the e-Business Technology Laboratory was established [1] to conduct a series of experimental projects.

Based on the results, the ebXML framework was adopted as the most promising.

The first strategic step is to host an implementation of the Registry at Agrokor, which would initially serve as the central point of integration among the many Agrokor's affiliates and partners. In the next step, it would attract other Croatian businesses to join. For this project, the SourceForge's open-source code base was used as the starting point.

As the Registry implementation effort advances, a need has emerged for development an implementation of an ebXML Registry Client. In this paper, the implementation details of the Registry Client are presented.

Section II describes components of the ebXML specifications vital for the understanding of the diagnostic Registry Client. Section III describes Registry Client implementation through its concepts and verification scenarios. A brief review of a JAXR API, a relatively new standardized API for accessing XML Registries is presented. The paper closes with Conclusion and References.

## II. COMPONENTS OF THE EBXML SPECIFICATION

The ebXML specification describes the following aspects of the e-business environment:

- the information model and service interface of the *ebXML Registry* and *Repository* [2], [3];
- structure, type and level of detail of business data contained in the *Collaboration Protocol Profile* (CPP), a document that describes a company's capabilities [4]. The company submits this document to the ebXML Repository, where it is publicly accessible;
- the procedure for partner discovery and subsequent negotiation of the collaboration protocol (resulting in the *Collaboration Protocol Agreement* – CPA) [4];
- the required information infrastructure that each company has to implement at its site in order to join the ebXML marketplace (*Business Service Interface*, *ebXML Message Service* [5]).

As mentioned above, the ebXML specification defines two types of data storage (see Fig. 1):

- *Repository* stores assorted business-related documents and other content available for public retrieval;

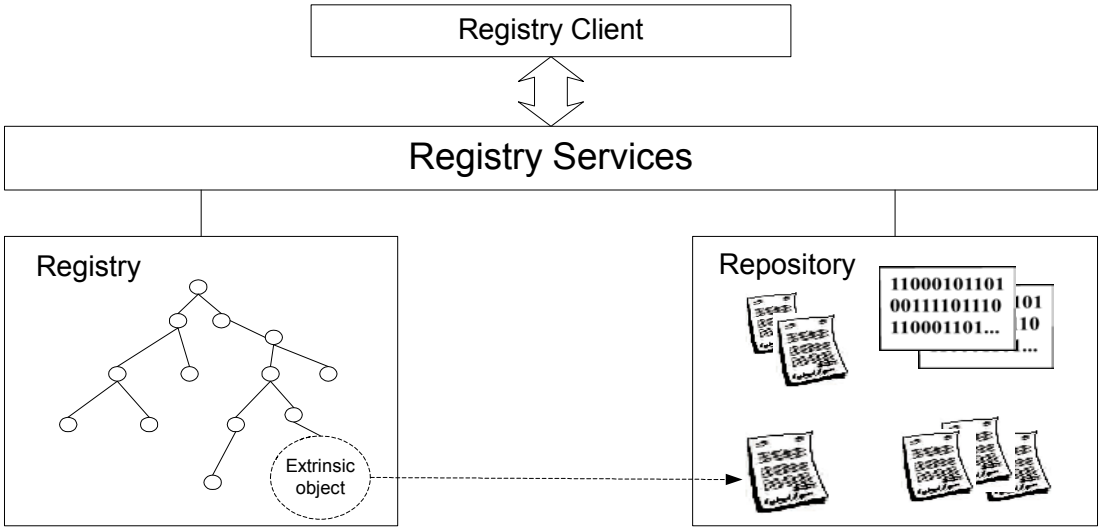


Figure 1. ebXML Registry/Repository high-level component overview

- *Registry* stores a network of descriptive, structured, searchable pieces of information about companies and contains links to the items in the Repository. Its organization is specified by the *Registry Information Model* (RIM).

Each item in the Repository is associated with a Registry object that represents it. This object is of the *ExtrinsicObject* class. The Repository item is always retrieved via its *ExtrinsicObject*. Typical items in the Repository include the above mentioned CPP documents, standardized CPA templates, business process descriptions, service interface descriptions, program code for accessing companies' business interfaces, etc.

### *A. Components of the Registry Services*

The ebXML Registry Services provide a public interface for maintenance, search, and retrieval of Registry objects and Repository items. They are functionally divided into two component interfaces:

- *Query Manager* (QM) offers several methods of searching the Registry, such as filter search, browse-and-drill-down, and SQL query. It also provides methods for retrieving Registry objects and their associated content from the Repository;

- *Lifecycle Manager* (LCM) provides the interface for the maintenance of Registry objects and Repository items, such as submitting, versioning, updating, superseding, deprecating, and removing.

Query Manager and Lifecycle Manager are defined as two distributed objects. The ebXML Registry Services Specification [2] specifies their public methods and properties.

### *B. Lifecycle of Repository Items*

Each Repository item has a lifecycle status assigned to it. The information about the status is contained in the Registry's *ExtrinsicObject* that represents it.

The ebXML specification defines the following lifecycle statuses:

- *Submitted*. The item is stored in the Repository, but is not yet available for use by business parties.

- *Approved*. The item is stored and available for use. This is the regular status of an active item. New Registry objects can be submitted that reference this item.

- *Deprecated*. The item is still in the Repository, but is outdated and due to be deleted. No new references can be made to it, but the existing ones are left intact.

- *Withdrawn*. The item has been removed from the Repository, but its associated *ExtrinsicObject* is still present, indicating this status. All references to the *ExtrinsicObject* are left intact. This object cannot be removed from the Registry until all references to it have been deleted first.

Our first step was to create an implementation of the Registry Client that would be able to carry out the first step, which is to *submit* items to the Repository and the accompanying metadata objects to the Registry. This means that the Client has to invoke the *submitObjects* method of the Registry's Lifecycle Manager (LCM).

## III. REGISTRY CLIENT IMPLEMENTATION DETAILS

### *A. Conformance Requirements*

In this subsection, the requirements that every ebXML-conformant Registry Client has to satisfy are discussed.

*1) Communication Bootstrapping:* Before the actual communication between a client and the Registry can commence, they have to go through a communication bootstrapping procedure. Through this procedure the client is made aware of the requirements it has to satisfy to communicate with the Registry. The critical aspect is the addressing information that the client has to discover. In our


ConTEL 2003, ISBN: 953-184-052-0

case, the client needs to find out the URI of the Registry's SOAP-over-HTTP interface. The supporting of this procedure is a requirement for a conforming Registry Client. In the case of the Client being used as a diagnostic tool for a local Registry, there is no need for an automatic discovery procedure. The addressing information is provided to the Client manually.

*2) RegistryClient Interface:* A Registry Client provides this object interface to the Registry Service so it can send back an asynchronous response. Asynchronous communication is needed when the Registry Service cannot immediately satisfy a request. The interface exposes only one method, *onResponse*. When the request is satisfied, the Registry Service invokes this method on the Client in order to deliver the response.

If there is a need for asynchronous communication, the Client is required to implement the *RegistryClient* interface. Currently, our implementation of the Registry does not call for asynchronous communication and thus the diagnostic Client does not implement this interface.

*3) Registry Services Schema:* Every request to the Registry Services is packaged into an XML document conforming to the Registry Services Interface Schema document, `rs.xsd`, which is a part of the Registry Services Specification. This document uses two other Schema documents, namely the RIM Schema, `rim.xsd`, and the Query Schema, `query.xsd`. These two documents are also a part of the Specification.

The main work on the diagnostic Client is concerned with preparing a conformant XML document for the *submitObjects* request.

### *B. Security Issues*

Security issues have not yet come into the focus of the ebXML Specification development. That does not imply that security has been omitted from the current version of the Specification. Rather, the framework for addressing all of the security issues has been laid out, but the working out of details has been left for a future version. Also, a minimum set of security provisions has already been specified.

The current status of these issues will be discussed briefly in this section.

*1) Content Integrity*—this has several aspects:

*a) Veracity.* The content should correlate precisely with the real-world facts. This aspect is very difficult to ascertain and this is not expected of a business Registry. Veracity is trusted upon the content's Submitting Organization.

*b) Source Integrity.* This aspect is crucial in forcing the Submitting Organization to commit to the veracity of its content. The ebXML Specification supports unambiguous identification of the Submitting Organization and the Responsible Organization. Under the currently supported scenarios, these two roles are always played by the same real-world entity.

*c) Data integrity.* The Registry can guarantee that the content it stores has not been tampered with and is the same content as submitted by the Submitting Organization. There are also mechanisms that enable the Client that received the content to verify that it is still tamper-free.

The Registry Services Specification extensively uses the XML Digital Signature technology for security. This technology provides a way to embed a digital signature within the XML document being signed. It also enables the sending of signer's certificate within the document.

*2) Authentication:* The Registry can authenticate the identity of the Principal associated with client requests. A Principal is any entity (a person, an organization, a software module) that can make requests to the Registry.

The identity of the Principal is authenticated by verifying the signature of the message header using the certificate of the Principal. The Registry must perform the authentication on a per-message basis. There is currently no concept of a session encompassing multiple messages. The support for sessions may be added in future versions of the Specification for optimization reasons.

In order to be able to get authenticated, a Principal must first register itself with the Registry as a Registry User. The Registry Services Specification does not currently specify the registration process itself, but it does specify the information model for storing data about Principal's identity, roles, groups, and the privileges associated with them.

In the implementation we use, a Principal is registered through a signed *submitObjects* request containing exactly one *User* object (and an unlimited number of other object types). If the signature is unknown to the Registry, it assumes that the *User* object contains information about a new Registry user. The user will be assigned the role of *ContentOwner* (see below) for the objects submitted in the request as well as any further objects submitted in future requests signed by the user.

*3) Authorization:* The identity of a Principal is associated with its access privileges. The privileges specify what type of operations a Principal is authorized to perform.

The Registry supports role-based privileges. In the current version, the Specification only supports several predefined roles. Future versions will support customized roles defined by Submitting Organizations. The roles are:

- *ContentOwner*. The Submitting Organization plays this role. It has full privileges on the content it owns and read-only privileges on all other content.
- *RegistryAdministrator*. This role has full privileges on all content.
- *RegistryGuest*. This role is played by a third-party Principal accessing the Registry. It has read-only privileges on all content.

Any role except the *RegistryGuest* requires that the Principal be a registered *User* of the Registry.

This scenario assures authentication of the Principal, integrity of content source, and the integrity of content's data.

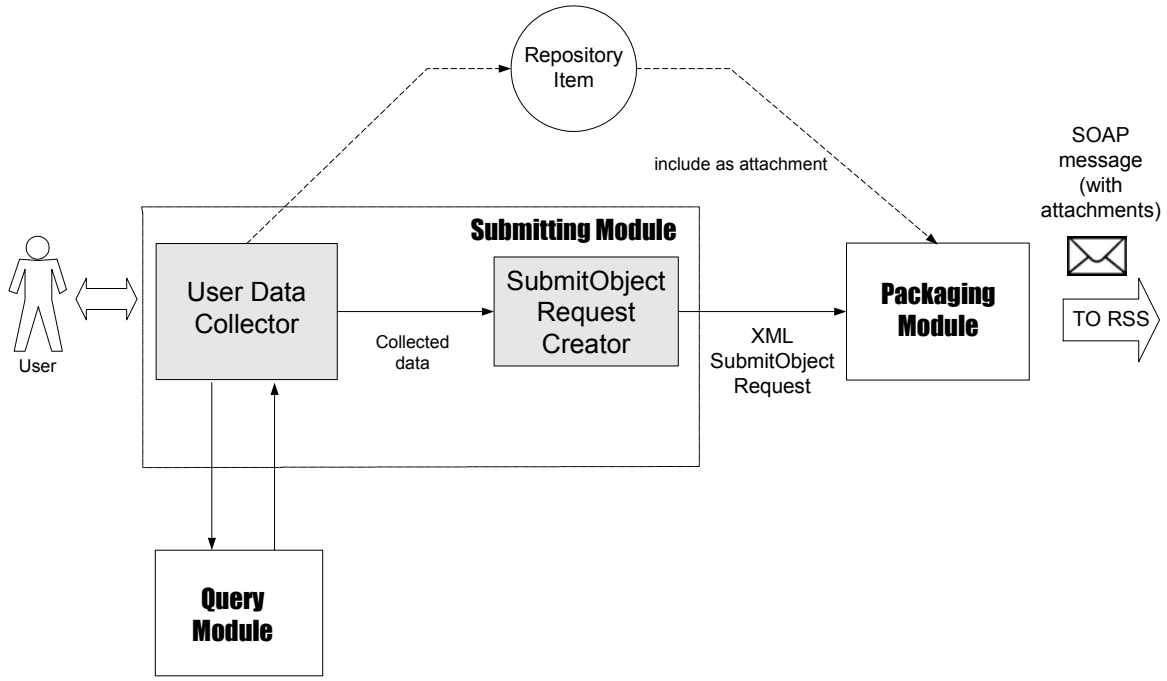


Figure 2. Workflow diagram of the submitObjects request preparation

Once the Principal's identity is known, this enables the Registry to check its privileges and decide whether it is allowed to perform the requested operation.

If the Client only needs read access to the Registry, it does not have to use any signatures. It will be assigned the RegistryGuest role.

*D. Content Classification*

One of the most important features of the data model of the Registry is content classification. It organizes the content into hierarchical categories which enable powerful searching techniques. Each object can be described by its place inside several taxonomy trees, like geography, industry branch, etc.

The Registry Information Model distinguishes between two technically different, but logically equivalent types of classification: external and internal. External classification relies on classification schemes not known to the Registry. The Registry contains only an identifier that locates the object inside such a scheme.

Internal classification locates the object inside a classification scheme tree that is stored in the Registry. This tree is composed of *ClassificationNode* objects. Each node can have only one parent node, but can have an unlimited number of child nodes. The Registry Information Model supports an unlimited number of internal classification schemes. A client can submit new schemes or add nodes to existing schemes.

To enhance the semantic content of classification, the classification of an object can be characterized by another classification. For example, a geographical classification of a company can designate the location of its headquarters, but also the location to which it ships its products. The context of the geographic classification is resolved by classifying its *Classification* object (see below) using a scheme that contains such nodes as "isLocatedIn," "shipsTo".

*E. Building the Request Document*

A *submitObjects* request is contained in an XML document with the root element named *SubmitObjectsRequest*. Its only subelement is the *LeafRegistryObjectList*, containing all the Registry objects being submitted. The content model of the latter element is specified by the Registry Information Model Schema. Each element in this list corresponds to a Registry object of the same-named class.

These are some of the Registry objects that can be submitted (class names are in UpperCamelCase):

- *Organization* object holds information about an organization on the ebXML marketplace. It is usually the central object type to which other objects are associated.
- *User* object holds information about a Registry user. An *Organization* usually has a *User* linked to it as its primary contact.
- *ExtrinsicObject* represents a content item being submitted to the Repository. There is one ExtrinsicObject for each Repository item in the payload, and they have to be in the same order.
- *Association* objects primarily link an *Organization* with other objects and specify the semantic context of their relation (association type);


ConTEL 2003, ISBN: 953-184-052-0

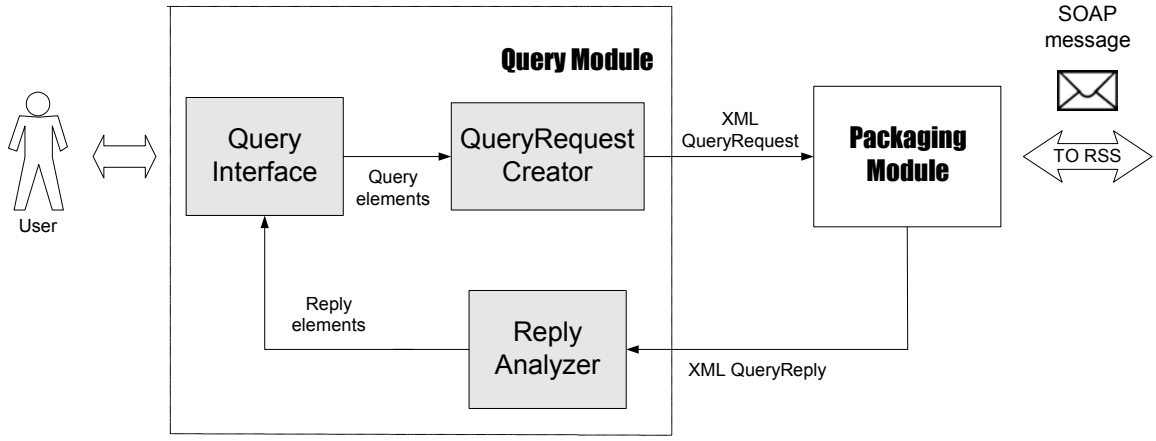


Figure 3. Workflow diagram of the Query request generation and result processing

- *Classification* objects classify another object by linking it to the internal *ClassificationNodes* or assigning it external classification ID's

- *ClassificationNode* objects extend the current classification schemes already in the Registry, or build new schemes;

Registry objects can be described by linking them to other objects via *Association* objects. An *Association* object contains two references, one for the source object, and one for the target. It also specifies the type of the association. For an *ExtrinsicObject*, typical association types are "RelatedTo," "Supersedes," "Contains," etc. The association is directed, with a clear distinction of source and target.

The most important description mechanism is classification. It is accomplished using the *Classification* object. Conceptually, it is a special form of association, but technically it is separate from it. The *Classification* object also contains source and target references, but the source is always a *ClassificationNode*. The target is the object being classified by this node. In the case of external classification, the source is not a reference to a *ClassificationNode* object, but an identifier valid in the external classification scheme.

All Registry objects are referenced by their Registry-specific ID's. An issue arises with the ID's of objects just being submitted. The Registry uses Universally Unique ID's (UUID) and these can be generated at the Client's side. But this is not a requirement. The Client can also use provisional ID's that are only unique within the single request document. The Registry will generate UUID's before committing the objects to its store. It is worth noting that although a UUID is unique in the sense that there cannot be two identical UUID's for different objects, a single object may still have multiple UUID's used in different contexts. This makes the notion of Registry-specific UUID's meaningful and important.

The newly submitted associations and classifications can reference objects already in the Registry. This is most typical of *Classification* objects using internal *ClassificationNodes*. A technical problem arises regarding how to discover the ID's of these objects. The proper way is to get the ID's through the Query Manager interface. This means that even if the diagnostic Client is designed to enable only the submitting of objects, it still requires a module that generates requests to the Query Manager interface.

For each Registry object being referenced in the Submit objects request, an *ObjectRef* element has to be provided in the XML document of the request. *ObjectRef* is a special element because it does not correspond to an object being submitted, but to an object already in the Registry. It contains the Registry-specific ID of that object.

### *F. Packaging the request*

The final form of a request to the Registry Services is a SOAP Message with Attachments. It is structured in compliance with the *SOAP Messages with Attachments* specification [5], [6]. It is transport protocol-independent and can be exchanged using any transport protocol, such as HTTP, FTP or SMTP. Our Registry Services use HTTP, which is the preferred protocol with respect to security and efficiency issues. The procedure of embedding the SOAP message inside an HTTP request is referred to as *transport binding*.

SOAP follows the peer-to-peer communication paradigm, so there is no inherent role distinction between the communicating parties. On the other hand, HTTP follows the client/server paradigm and always involves a request-response pair of messages. This pair is used for the synchronous message exchange pattern in SOAP. For each *submitObjects* request message sent inside an HTTP request, the *RegistryResponse* message is sent back inside the corresponding HTTP response.

A SOAP Message with Attachments is a MIME/Multipart *Message Package* containing the following MIME parts:

- the *Header Container*, containing a SOAP 1.1 compliant message, referred to as the *SOAP Message.* It is the central element of an ebXML Message;

- zero or more *Payload Containers,* containing, in this case, the Repository Items being submitted. These are the SOAP Message *attachments.*

The *SOAP Message* is an XML document that consists of the SOAP Envelope root element. This element consists of the following subelements:

- a *Header* element, specifying the nature of the ebXML message (such as whether the message is a request or an answer; the type of request/answer, etc.). This element also contains the digital signature and optionally the certificate;

- a *Body* element, containing the main content of the message, in our case the *submitObjects* request document.

In our implementation, we have used the Java API for XML Registries (JAXR) technology discussed below to generate requests to and interpret responses from the Registry.

### *G. A Use Case Scenario*

The scenario of a typical use case can be followed in Fig. 2. A user has prepared data about his organization and wishes to submit it to the Registry. He has to provide information such as name, description, postal and e-mail addresses, etc. He can also provide external links relevant to the organization (e.g. URI's of relevant Web resources) and/or external identifiers of the organization that are meaningful outside the context of the Registry (such as stock exchange tickers). The Client application makes this easy by presenting forms where the necessary information can be filled in.

The central aspect of information about the organization is its classification. If the organization is not properly classified, it will probably not get discovered by parties interested in it. In order to find the classification nodes that best describe the organization, the Registry's classification schemes can be browsed. This is accomplished by interacting with the Query module as shown in Fig. 3. The Client application downloads a classification tree from the Registry and presents it in a graphical form to the User. The tree is visualized similar to a file system's folder structure. The preferred method of working with classification trees is to request from the Registry only the nodes that actually have to be displayed. This is because an entire classification tree can be very large and usually only a small part of it will actually be requested by the user.

When all data has been gathered, the request document is created. This document is packaged along with any Repository items into a SOAP Message with Attachments. The message is then sent to the Registry using the transport binding layer. The Registry returns a response, notifying the user whether the request has been successfully fulfilled.

### *H. Java API for XML Registries (JAXR)*

Java API for XML Registries [7] is a standardized API designed to be used to access a variety of XML-based business registries. It is still under development and has reached the Final Proposed Draft phase. The API is specified in a white paper accompanied by the definition of Java interfaces. To use the API, a JAXR *provider* layer has to be developed. This layer implements the functionaliry of the interfaces defined by the API layer. A JAXR provider is specific to a particular type of registry.

By using JAXR, the Registry Client's functionality is systematically split into:

- a client layer that uses the JAXR API;

- a provider layer that implements JAXR for the ebXML Registry.

This approach could prove especially valuable in the context of still unstable ebXML specifications. A lot of changes to the specifications will require intervention only to to the provider layer. Also, this makes it easy to develop a registry client that can access different types of registries by just plugging in the appropriate JAXR providers.

## IV. CONCLUSION

As a part of a project named "Networked Economy," backed up by Croatian Ministry of Science and Technology, at the Department of Telecommunications, Faculty of Electrical Engineering and Computing, University of Zagreb there is research in progress on e-Business-related projects. One of the projects is concentrated on implementing the ebXML Registry. As the Registry implementation effort advances, a need has emerged for the existence of the ebXML Client to maintain and debug the system under development. In this paper, an implementation of a lightweight Registry Client application that can meet this requirement was presented.

Future work on this application will be done concerning the development of new features of the Registry Service. This pertains especially to the security features that will be specified in future versions of the ebXML Registry Services Specification.

## REFERENCES


[1] M. Topolnik, D. Pintar, M. Sokic: Experimental Implementation of Emerging e-Business Technologies: EbXML and PKI, Proceedings of the Joint Tutorials and Conference MIPRO2002 – Electronic Commerce, Opatija, 2002, pp. 1-6.

[2] OASIS/ebXML Registry Technical Committee, OASIS/ebXML Registry Services Specification, http://www.ebxml.org/specs/ebrs2.pdf, December 2001.

[3] OASIS/ebXML Registry Technical Committee, OASIS/ebXML Registry Information Model v2.0, http://www.ebxml.org/specs/ebrim2.pdf, December 2001.

[4] Trading Partners Team, Collaboration-Protocol Profile and Agreement Specification v1.0, http://www.ebxml.org/specs/ebCCP.pdf, May 2001.

[5] Transport, Routing & Packaging Team, Message Service Specification v1.0, http://www.ebxml.org/specs/ebMS.pdf, May 2001.

[6] J. J. Barton, S. Thate, Henrik F. Nielsen, SOAP Messages with Attachments, http://www.w3.org/TR/SOAP-attachments, Microsoft, October 2000.

[7] Sun Microsystems, Java API for XML Registries v0.9, Proposed Final Draft, http://java.sun.com/xml/downloads/archive-jaxr.html, February 2002.