JAebXR: a Java API for ebXML Registries for Federated Health Information Systems

Antonio Messina, Pietro Storniolo and Alfonso Urso
ICAR - CNR
Palermo, Italy
Email: {messina, storniolo, urso}@pa.icar.cnr.it

***Abstract*—The traditional Java API for Registries (JAXR) provides a useful way for Java developers to use a single simple abstraction API to access a variety of registries. The unified JAXR information model (Infomodel), which describes content and metadata within registries, provides a simple way to access registries information. However, when a JAXR registry provider implements the OASIS (Organization for the Advancement of Structured Information)/ebXML Registry Services Specification, it internally works with ebXML Registry Information Model (RIM) objects which should be exposed as JAXR Infomodel objects. Furthermore, any application using ebXML RIM objects, before the access to the registry via a JAXR provider, has to convert them to the Infomodel counterpart. To deal with this problem, in this paper we suggest a new tool for the ebXML software development, which focuses on an extension of the traditional JAXR layer, providing the native use of ebXML RIM objects in client side applications with a direct access to the ebXML RIM SOAP service, and avoiding any conversion to/from JAXR Infomodel objects. The proposed approach reduces the developers duty, allowing them to focus exclusively on the ebXML objects use and it considerably speeds up the interactions with every ebXML registries.**

Keywords–ebXML, Registry Services, Registry Client, SOAP, JAXR, JAVA.

I. INTRODUCTION

In recent years, we have been involved in the development of some software components within the project *OpenInFSE*, an experimental interoperability infrastructure based on the InFSE architecture [1], which represents a multi-level service-oriented architecture for sharing medical data among federated *Health Information Systems* (HIS) [2]. This infrastructure is based on the extensive use of Web Service technology and XML data exchanged as Simple Object Access Protocol (SOAP) messages in accordance to the Health Level 7 (HL7) [3] Clinical Document Architecture (CDA) standard.

All of the software components included in the InFSE *Component layer* interact with the *Federated Index Registry*, which enables the query of medical data, managed by several HIS, to a federated system of regional registries in Italy, each of them able to localize the data achieved in the regional repositories.

Currently, there are two preeminent registry standards: Universal Description, Discovery, and Integration (UDDI) [4] and Electronic Business using eXtensible Markup Language (ebXML) [5] [6]. Using one of these registry standards, it is possible to publish a set of Web services, enabling internal or external business partners to discover them.

A registry typically works as electronic Yellow Pages, where information about businesses, and the products and services they offer are published and discovered. A registry can also serve as a database or as a storage of shared informations.

A registry can also work as an electronic bulletin board in which the partners share information in a dynamic and ad hoc manner.

Submission and storing of shared information are important operations performed by registry-service clients. These clients also need to complete various registry management operations, such as identifying, naming, describing, classifying, associating, grouping, and annotating registry metadata. Finally, clients also must be able to query, discover, and retrieve shared information from the registry so that they expect that a typical registry supports most of these operations.

The ebXML is designed to create a global electronic market place where enterprises of any size, anywhere, can find each other electronically and conduct business using exchange of XML messages according to standard business process sequences and mutually agreed trading partner protocol agreements. This standard overcomes the limitations of existing business-to-business frameworks and technologies, because, for example, UDDI does not provide repository capability for business objects, SOAP in its basic form does not provide reliable and secure message delivery, and the Web Service Definition Language (WSDL) does not address business collaboration.

The Java API for XML Registries (JAXR) [7] provides a standard *Application Programming Interface* (API) for publication and discovery of Web services through underlying registries. JAXR does not define a new registry standard. Instead, this standard Java API performs registry operations over a various set of registries and defines a unified information model to describe registry contents.

The JAXR specification defines a general-purpose API, allowing any JAXR client to access and interoperate with any business registry accessible via a JAXR provider. In this sense, JAXR provides a Write Once, Run Anywhere API for registry operations, simplifying Web services development, integration, and portability.

During the development of OpenInFSE project, JAXR has been initially adopted to access to underlying ebXML registries. Even though JAXR can be used for the access to ebXML registries, it can be considered unsuitable because it heavily depends on the registry implementation, it involves too many data conversion and it may lead to a serious general inefficiency of the entire infrastructure.

 ISBN: 978-1-61208-408-4

In this paper, the development of JAebXR, which is a JAXR extension, is proposed. This extension can be viewed as a complementary API to offer an ebXML provider implementation and to complete the current JAXR reference implementation.

The paper is organized as follow. Section 2 presents the JAXR architecture, also illustrating some typical registry operation. Section 3 presents the motivations for this work. Section 4 presents our proposed approach to assist the ebXML client code development. Section 5 introduces the open source ebXML registry service developed as natural counterpart of JAebXR within the OpenInFSE Project. Finally, conclusions and the presentation of some future works are reported.

II. JAXR ARCHITECTURE

The JAXR architecture defines three important architectural actors:

- A registry provider implements an existing registry standard, such as the Organization for the Advancement of Structured Information (OASIS)/ebXML Registry Services Specification 3.0.
- A JAXR provider offers an implementation of the JAXR specification approved by the Java Community Process (JCP) in May 2002. You can implement a JAXR provider as its own JAXR-compliant registry provider. However, you would more likely implement a JAXR provider as an interface to an existing registry provider, like UDDI or ebXML type. Currently, the JAXR reference implementation 1.0 offers a JAXR UDDI provider implementation.
- A JAXR client is a Java program that uses JAXR to access the registry provider via a JAXR provider. A JAXR client can be either a standalone Java 2 Platform Standard Edition (J2SE) application or Java 2 Platform Enterprise Edition (J2EE) components, such as Enterprise JavaBeans (EJBs), Java Servlets, or JavaServer Pages (JSs). The JAXR reference implementation also supplies one form of a JAXR client, a Swing-based registry browser application.

Because JAXR offers a standard API for accessing various registry providers and a unified information model to describe registry contents, JAXR clients, whether HTML browsers, J2EE components, or standalone J2SE applications, can uniformly perform registry operations over various registry providers.

Figure 1 shows a high-level view of the current JAXR architecture.

Note that the JAXR client connects with the JAXR provider, not the registry provider. The JAXR provider acts as a proxy on the client's behalf, directing and invoking methods on the appropriate registry provider. The connection maintains client state. In addition, the JAXR client dynamically sets its authentication information and communication preference on the connection any time during the connection's lifetime.

After the JAXR client invokes JAXR capability-level methods, the JAXR provider transforms these methods into registry-specific methods and executes requests to the underlying registry providers.

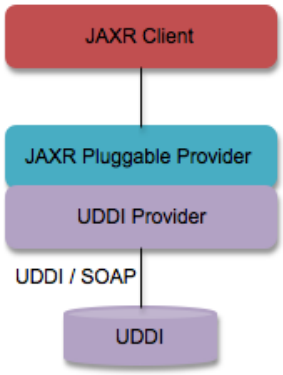

Figure 1. Current available JAXR architecture.

The communication protocol between a JAXR provider and a registry provider depends on the registry type and it is transparent to the JAXR client. A JAXR provider communicates with the UDDI registry provider by exchanging basic SOAP messages, while the JAXR provider should communicate with an ebXML registry provider through SOAP messaging or ebXML message service.

A. Capability profiles

Because among registry provider capabilities some diversity exists, the JAXR expert group decided to add multilayer API abstractions through capability profiles. A capability level is assigned to each method of a JAXR interface, and those JAXR methods with the same capability level define the JAXR provider capability profile.

Currently, JAXR defines only two capability profiles: level 0 profile for basic features and level 1 profile for advanced features. Level 0's basic features support the so-called business-focused APIs, while level 1's advanced features support generic APIs. At the minimum, all JAXR providers must implement a level 0 profile. A JAXR client application using only those methods of the level 0 profile can access any JAXR provider in a portable manner. JAXR providers for UDDI must be level 0 compliant.

JAXR providers can optionally support the level 1 profile. The methods assigned to this profile provide more advanced registry capabilities needed by more demanding JAXR clients. Support for the level 1 profile also implies full support for the level 0 profile. JAXR providers for ebXML must be level 1 compliant.

B. JAXR information model

Invoking life-cycle and query management methods on the JAXR provider requires the JAXR client to create and use the JAXR information model objects.

The JAXR information model resembles the one defined in the ebXML Registry Information Model 2.0, but also accommodates the data types defined in the UDDI Data Structure Specification.

Although developers familiar with the UDDI information model might face a slight learning curve, once understood, the JAXR information model will provide a more intuitive and natural interface to most developers.

 ISBN: 978-1-61208-408-4

C. JAXR interations

Figure 2 shows the JAXR interactions in a client-server communication.

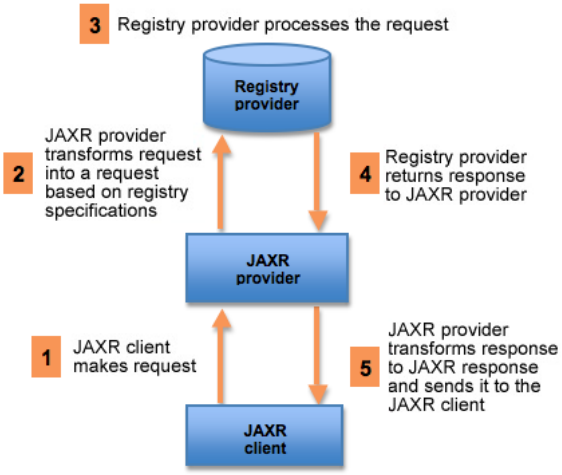

Figure 2. JAXR interactions.

1) A JAXR client uses JAXR interfaces and classes to request access to a registry. The client sends the request to a JAXR provider.
2) When a JAXR provider receives a request from a JAXR client, it transforms the request into an equivalent request that is based on the specifications of the target registry. The JAXR provider then passes this transformed request to a registry provider.
3) The registry provider receives a request from a JAXR provider and processes it. The process is then reversed.
4) The registry provider returns a response to the JAXR provider, which transforms it to an equivalent JAXR response.
5) The JAXR provider sends the JAXR response to the JAXR client.

III. MOTIVATION

In the development of applications or software layers that need to interact with ebXML registries there are some alternative ways:

1) direct use of the ebXML SOAP service exposed by a standard ebXML registry;
2) direct use of registries proprietary APIs;
3) JAXR APIs.

Because the first is well documented but not very practical and a developer should try to avoid the second for portability reasons, JAXR seems to be an obvious choice.

JAXR was our first choice but immediately we have obtained unsatisfactory results for the following reasons:

- Both the request from a JAXR client to a JAXR provider and the JAXR response sent back to the JAXR client carry Infomodel objects. It means that the JAXR client has to *manage* such objects.
- As mentioned in previous section, the current JAXR reference implementation offers only a JAXR UDDI provider implementation. It means that when we develop some ebXML-related piece of software, we need to map our ebXML objects to JAXR Infomodel objects for the requests and viceversa for the responses.
- Avaliable open-source ebXML registry providers, i.e., Omar [8], present a JAXR side-server interface but they internally work on ebXML objects. It means that external JAXR requests are translated to ebXML requests and the ebXML responses are translated back to JAXR responses.

Figure 3 shows the ebXML-over-JAXR interactions in a client-server communication.

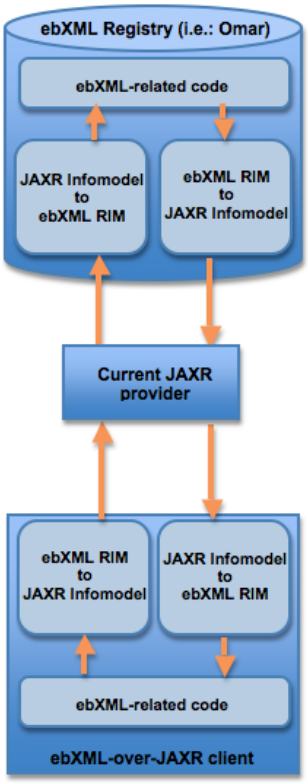

Figure 3. ebXML-over-JAXR interactions.

We refer interested readers to [9] for more details.

The reasons stated above involve considerable complexity in the code development and a substantial performance degradation.

In the following section, we propose an extension of the traditional JAXR layer, providing the native use of ebXML RIM objects in client side applications with a direct access to the ebXML RIM SOAP service, and avoiding any conversion to/from JAXR Infomodel objects.

The proposed approach reduces the developer's duty, allowing them to focus exclusively on the ebXML object's use and it may considerably speed up the interactions with every ebXML registries.

 ISBN: 978-1-61208-408-4

IV. JAXR EXTENSION: JAEBXR

The Java API for ebXML Registries (JAebXR) library has been developed in order to facilitate, standardize and optimize interactions with ebXML registries.

Its architecture is directly derived from JAXR, incorporating and extending its functionality to ensure support to the types of objects and services defined by OASIS ebXML RegRep 3.0 specifications.

For this reason JAebXR can be considered as a JAXR Provider Level 1 implementation, because it also supports UDDI registries type (level 0).

Figure 4 shows a high-level view of the JAebXR architecture.

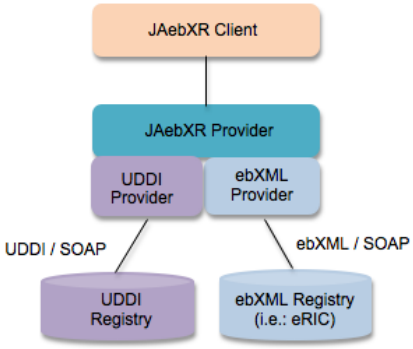

Figure 4. JAebXR architecture.

The API is independent of the particular implementations of ebXML registries. Interactions with ebXML registries occur, in fact, simply by invoking the SOAP web services exposed, as per OASIS specifications. In such circumstances, queries and transactions exclusively refer to ebXML RIM objects, properly encapsulated using the Java Architecture for XML Binding (JAXB) [10].

To speed up operations, in particular queries on registry objects, the API uses the in-memory object cache implementation provided by *cache2k* [11].

A. Implementation

The package *javax.ebxml.registry*, like the package *javax.xml.registry* in JAXR, contains the definition and implementation of the interfaces to register access, as shown in Figure 5.

The classes *BusinessLifeCycleManager*, *BusinessQueryManager*, *Connection*, *ConnectionFactory*, *DeclarativeQueryManager*, *LifeCycleManager*, *QueryManager*, and *RegistryService* are implementations of the homonymous JAXR interface classes, extended by specific methods for the ebXML RIM objects.

The sub packages *javax.ebxml.registry.security* and *javax.ebxml.registry.soap* handle the secure client-server SOAP interactions using Apache WSS4J [12], which implements the primary security standards for web services, namely the OASIS Web Services Security (WS-Security) specifications [13].

In the following subsections, a more detailed description of some of the main classes is reported.

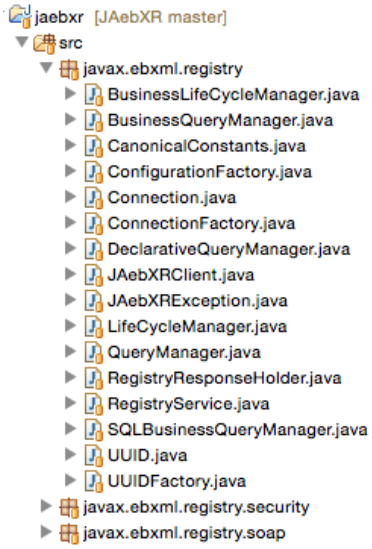

Figure 5. The JAebXR Java package.

B. ConfigurationFactory

Configuration issues are managed by the *ConfigurationFactory* class, which was built as a singleton: there is just a private constructor and a static getter method that returns an instance of the class, creating it in advance or at the first invocation of the method, by storing the reference in a static private attribute.

The class expects configuration directives given in a text properties file called *ebxml.properties* and, once instantiated, reads that file to set the required attributes to instantiate a *SOAPMessenger* object to invoke the ebXML registry web service in authenticated mode.

Figure 6 shows a sample configuration file:

```
connectionFactoryClass = it.cnr.icar.eric.client.
    xml.registry.ConnectionFactoryImpl

aliasName = urn:freebxml:registry:predefinedusers:
    registryoperator
aliasPass = urn:freebxml:registry:predefinedusers:
    registryoperator

keystorePath =
    /opt/eric/3.2/data/security/keystore.jks
keystorePass = ebxmlrr

certificateType = JKS

registryURL =
    http://localhost:8080/eric/registry/soap

cacheExpirationTime = 15
cacheMaxSize = 500

sqlQueries = false
```

Figure 6. Sample ebxml.properties

The supported attributes are:

- *connectionFactoryClass*: fully qualified name of the registry class, which implements the JAXR ConnectionFactory interface. It is required only when JAebXR

is used in a JAXR-way, as better specified in the next subsection;

- *aliasName* and *aliasPass*: registry user credential;
- *keystorePath*, *keystorePass* and *certificateType*: keystore related parameters to access to the user certificate;
- *cacheExpirationTime*: time duration in minutes after an entry expires. A value of 0 disables the cache;
- *cacheMaxSize*: maximum cache size limit in KBytes;
- *sqlQueries*: enable the use of optional SQL-92 [14] queries instead of standard ebXML filter queries.

C. ConnectionFactory

The *ConnectionFactory* class is the base class essential for the creation of a JAXR connection, which can be done by searching via the Java Naming and Directory Interface (JNDI) [15] or directly, through the use of the static method *newInstance()*.

We have chosen the second way, which requires the instantiation of the class that implements the interface *ConnectionFactory*.

The registry connection is also completely managed in terms of authentication, using the user credentials contained in a specific keystore. That means that an invocation of the method *createConnection()* returns a ready-to-use *Connection* object.

D. LifeCycleManager and BusinessLifeCycleManager

The LifeCycleManager interface class in JAXR is devoted essentially to the management of Infomodel objects by the declaration of several abstract methods.

The LifeCycleManager class in JAebXR fully implements the interface javax.xml.registry.LifeCycleManager and extends it to support all the ebXML RIM objects by methods named using the *Type* suffix. It means, for example, that the creation of a RIM registry object is realized by the *createRegistryObjectType()* method, the creation of a RIM classification scheme is realized by the *createClassificationSchemeType()* method, and so on.

Object storage and cancellation are achieved by the methods *saveObjectTypes()* and *deleteObjectTypes()*. Both of these methods call the generic method *submitObjectTypes()*, which actually implements the above operations using SOAP messages sent directly to the ebXML registry.

The BusinessLifeCycleManager class extends LifeCycleManager and implements the ebXML version of traditional JAXR methods.

E. RegistryService

The RegistryService class is the principal interface implemented by a JAXR provider and it can be obtained from a Connection to a registry.

In JAebXR, a RegistryService object is instantiated even when there is no JAXR connection. Therefore you can always use it to obtain the references to the fundamental BusinessLifeCycleManager, DeclarativeQueryManager and BusinessQueryManager objects.

F. DeclarativeQueryManager

The class fully implements the interface *javax.xml.registry.DeclarativeQueryManager* and extends it to support all the ebXML query types, such as:

- ebXML Registy Services Filter queries;
- SQL-92 queries;
- Stored queries;

The queries are incapsulated in an *AdhocQueryRequest* object before they are sent via SOAP. Then the results are returned as a complex type *RegistryResponseType* object.

G. BusinessQueryManager and SQLBusinessQueryManager

The BusinessQueryManager class, which is exposed by the Registry Service, implements the business style query interface. It supports all the standard JAXR methods and it also provides several new methods to interact with ebXML registries. Queries are done using the mandatory ebXML query types.

The derived class SQLBusinessQueryManager implements some of the business queries rewriting them using SQL-92, optionally supported by the OASIS ebXML standards.

H. JAebXRClient

This is an auxiliary class, designed as a singleton, provided to develop ebXML clients simply and quickly. Once instantiated, it immediately sets up all that is necessary and it creates all the main objects able to interact with an ebXML registry:

- *lcm*: BusinessLifeCycleManager;
- *dqm*: DeclarativeQueryManager;
- *bqm*: BusinessQueryManager (or SQLBusinessQueryManager, according to the sqlQueries parameter's value in configuration file).

Their values are available to the outside via the getter methods *getBusinessLifeCycleManager()*, *getDeclarativeQueryManager()* and *getBusinessQueryManager()*, as shown in the figure below.

```
public class ClientTest {

   JAebXRClient ebc = JAebXRClient.getInstance();
   BusinessLifeCycleManager lcm =
       ebc.getLifeCycleManager();
   BusinessQueryManager bqm =
       ebc.getBusinessQueryManager();
   DeclarativeQueryManager dqm =
       ebc.getDeclarativeQueryManager();

   public ClientTest() {
     ...
   }

   ...
}
```

Figure 7. JAebXRClient sample use

V. TEST CASE

The use of the library is pretty simple: first of all the configuration file is prepared and then, as reported in the sample code shown in Figure 7, an instance of JAebXRClient class is created.

All our tests have been done using the *ebXML Registry by ICAR CNR* (eRIC) version 3.2 [16].

To test the ebXML specifications compliance, we have chosen the eRIC JAXR client test suite [17] as starting point and we have rewritten it in a JAebXR way. The library passed all the tests.

Finally, we have put together various pieces of code from previous tests to execute a sort of benchmark to also check the performances. We did the same with the JAXR version of the source code.

We have grouped the tests in the following four macro-categories:

- RIM
  - creation, addition of attributes, storage, load and deletion of a Person object and a User object;
  - creation, storage, load and deletion of ClassificationScheme object and of some its multi-level ClassificationNode childs;
  - creation, storage, modification, load and deletion of a RegistryPackage and associated Classification;
  - nested Classifications update;
- ebXML:
  - creation and deletion of an Association between two objects;
  - creation, storage and deletion of an ExtrinsicObject;
  - simple AdhocQuery execution;
- BusinessQueryManager:
  - search of services by organization, by name pattern, by organization and name;
  - creation of a ServiceBinding with a specification object that is a ClassificationNode with no parent and then search of a service bindings by specification object;
- LifeCycleManager:
  - test of SetStatusOnObject extension protocol, which allows setting status of an object to any ClassificationNode within canonical StatusType ClassificationScheme;
  - invocations of various API methods with null parameter;
  - implicit storage of true-composed objects;
  - implicit storage of associations and associated objects.

Figure 8 shows the tests results, where execution times are in milliseconds. Also note that the JAebXR version was executed two times: one with the configuration parameter *sqlQueries* set to *false*, the other with a value of *true*.

If you take into account that the JAebXR code is actually just a research project and it isn't ready for production use, tests results can be considered very interesting, namely:

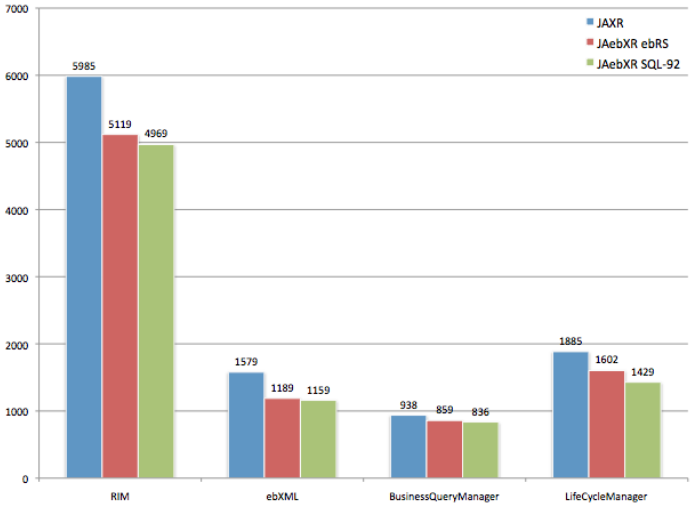

Figure 8. JAebXR Benchmarks

- with standard ebXML filter queries, we saw an increase in performance equal to 8.42% minimum to a maximum equal to 24.69%;
- with SQL-92 mode enabled, the improvement was even more relevant: minimum equal to 10.87% and maximum equal to 26,59%.

VI. CONCLUSION AND FUTURE WORK

In this work, we have highlighted the constraints and challenges faced by developers for producing Java code able to handle ebXML RIM objects in a JAXR environment.

We have explained how to use the JAXR API and methods of implementation of related source code.

Lack of an efficient handling of ebXML RIM objects was our motivation for the design and implementation of a new assistance layer for a better use of standard ebXML registriy services.

The result of current work is a simple easy-to-use API which extends JAXR to manage ebXML objects in a efficient way. Moreover, although the main scope of this work was related to a HIS project, the API is absolutely independent of health information.

Future work to improve this layer, along with the eRIC Registry, includes full support of the latest ebXML specifications published by the OASIS Consortium, ebXML RegRep v4.0 [18].

REFERENCES

[1] M. Ciampi, G. De Pietro, C. Esposito, M. Sicuranza, and P. Donzelli, "On Federating Health Information Systems," International Conference on Green and Ubiquitous Technology, Jul. 2012, pp. 139–143, ISBN: 978-1-4577-2172-4.

[2] R. Hauxe, "Health information systems," International Journal of Medical Informatics, vol. 75(3), Mar. 2012, pp. 268–281.

[3] D. F. Sittig, G. J. Kuperman, and J. M. Teich, "WWW-based interfaces to clinical information systems: the state of the art," Proceedings of the AMIA Annual Fall Symposium, 1996, pp. 694–698.

[4] OASIS UDDI Specification Technical Committee, "Universal Description, Discovery and Integration v3.0.2 (UDDI)," 2004, URL: https://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm [accessed: 2015-01-22].

[5] OASIS ebXML Registry Technical Committee, “ebXML Registry Information Model (RIM) v3.0,” 2005, URL: http://docs.oasis-open.org/regrep/regrep-rim/v3.0/regrep-rim-3.0-os.pdf [accessed: 2015-01-22].

[6] OASIS ebXML Registry Technical Committee, “ebXML Registry Services and Protocols v3.0,” 2005, URL: http://docs.oasis-open.org/regrep/regrep-rs/v3.0/regrep-rs-3.0-os.pdf [accessed: 2015-01-22].

[7] Java Community Process, “JSR 93: Java API for XML Registries 1.0 (JAXR),” 2002, URL: https://jcp.org/ja/jsr/detail?id=93 [accessed: 2015-01-22].

[8] “Omar: OASIS ebXML Registry Reference Implementation Project (ebxmlrr),” 2007, URL: http://ebxmlrr.sourceforge.net [accessed: 2015-01-22].

[9] M. Topolnik, D. Pintar and I. Matasic, “Implementation of the ebXML Registry Client for the ebXML Registry Services,” Proceedings of the 7th International Conference on Telecommunication, Zagreb, Croatia, Jun. 2003, pp. 551–556, ISBN: 953-184-052-0.

[10] Java Community Process, “JSR 222: Java Architecture for XML Binding (JAXB) 2.0,” 2009, URL: https://jcp.org/en/jsr/detail?id=222 [accessed: 2015-01-22].

[11] headissue GmbH, “cache2k - High Performance Java Caching,” 2015, URL: http://cache2k.org [accessed: 2015-03-19].

[12] Apache Software Foundation, “Apache WSS4J - Web Services Security for Java,” 2015, URL: https://ws.apache.org/wss4j/ [accessed: 2015-03-19].

[13] OASIS Web Services Security Technical Committee, “WS-Security OASIS Standard 1.1,” 2006, URL: https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf [accessed: 2015-03-19].

[14] ISO ANSI, “Database Language SQL ISO/IEC 9075:1992,” , Jul. 1992.

[15] Oracle Corporation, “Java Naming and Directory Interface 1.2,” 1999, URL: http://www.oracle.com/technetwork/java/jndi-150206.pdf [accessed: 2015-01-22].

[16] A. Messina, “eRIC: ebXML Registry by ICAR CNR,” 2014, URL: https://github.com/IcarPA-TBlab/eRIC [accessed: 2015-03-02].

[17] A. Messina, “eRIC Test suite,” 2014, URL: https://github.com/IcarPA-TBlab/eRIC/tree/master/eric-test-3.2 [accessed: 2015-03-02].

[18] OASIS ebXML Registry Technical Committee, “ebXML RegRep v4.0,” 2012, URL: http://docs.oasis-open.org/regrep/regrep-core/v4.0/os/regrep-core-v4.0-os.zip [accessed: 2015-01-22].

 ISBN: 978-1-61208-408-4