Evaluation of a Document Oriented Resource Directory Performance

Stevan Jokić, Srdjan Krčo, Igor Dejanović, Jelena Vučković, Nenad Gligorić, and Dejan Drajić

***Abstract* — In this paper, two implementations of a Resource Directory based on document oriented databases are described. Resource Directory is a directory storing descriptions of resources available in an Internet of Things system. Their performances are compared with performance of a Resource Directory with the same functionality, but implemented using a relational, SQL database. The evaluation results show that the document based Resource Directories provide more flexible management of the resources and a shorter response time.**

***Keywords* —Document oriented databases, future Internet, IoT, native XML databases, relational databases, resources.**

I. Introduction

THE technological progress in the domain of low cost microprocessors and embedded devices with wireless communication capabilities is paving the way towards implementation of an Internet of Things (IoT). IoT will integrate numerous small devices embedded in the physical environment, the so called "things", and make it possible to access the information these devices provide in a uniform manner or to control the devices and actuate the environment around them. With further interaction and integration with other Internet services, this will form the basis for a Future Internet.

As the devices considered by IoT are resource constrained in terms of their capabilities (processing power, storage, power supply), there are ongoing efforts to design lightweight protocols suitable for such environment, but at the same time capable of providing Internet like experience. One of the important IoT components is a directory that keeps track of available resources in a given domain, the so called resource directory (RD). This directory stores standardized resource descriptions and makes it possible for other IoT components to search for required resources even if the resources are not available at that moment.

In this paper, the analysis of the performance of different databases when used to store descriptions of IoT resources is done.

The remainder of the paper is organized in the following manner. Section II describes the concept of RD in more details. Further usage of the RD is described in Section III. Section IV describes implementation of RD based on SEDNA database, while Section V describes implementation of the same functionality based on the Mongo database. Section VI provides information about the performance evaluation tests and results. The concluding remarks are given in Section VII.

II. Resource Directory

Resource Directory (RD) is originally designed and implemented in the FP7 SENSEI project [1]. All entities in a SENSEI system were considered as resources. Resource Descriptions are introduced as a human/machine understandable representation of the resources and are formatted following an agreed XML schema. Each Resource Description is represented as an XML structure that contains a set of tags describing the Resource, as well as the URL where the so called Resource Endpoint (REP) - access point to the Resource is located. A REP provides the Resource Access Interface (RAI) for accessing the Resource. The following XML is an example of a resource description for a gas sensor deployed on a bus in Pančevo as a part of the ekoBus system [4].

```
<Resource-Description>
  <Resource-ID>
urn:sensei:ericsson.com:EnvironmentalSensors:bu
sgps:358278006369805
  </Resource-ID>
  <Name>Bus Location Sensor</Name>
  <Tag>Bus</Tag>
  <Tag>GPS</Tag>
  <Tag>Sensor</Tag>
  <Tag>Tracker</Tag>
  <Tag>Pančevo</Tag>
  <Tag>24</Tag>
  <Tag>358278006369805</Tag>
  <RAI-Description>
    <Description>GET returns sensor values
    (RDF)</Description>
    <REP-Locator>
    http://www.ekobus.rs/rephandler/gps/358278
  006369805
    </REP-Locator>
  </RAI-Description>
</Resource-Description>
```

This paper describes work undertaken in the context of the SmartSantander project. The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° ICT-2009-257992.

This paper describes work partially undertaken in the context of the FP7 HOBNET project. The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 257466.

Stevan.J, Ericsson Serbia, University of Novi Sad, (e mail: stevan.jokic@ericsson.com).

Srđan. K.. Ericsson Serbia, University of Belgrade, e-mail: srdjan.krco@ericsson.com

Igor D. Ericsson Serbia, University of Novi Sad, (e mail: igor.dejanovic@ericsson.com).

Jelena V., Ericsson Serbia, e-mail: jelena.vuckovic@ericsson.com

Nenad G., Ericsson Serbia, University of Belgrade, e-mail: nenad.gligoric@ericsson.com

Drajić D., Ericsson Serbia, e-mail: dejan.drajic @ericsson.com

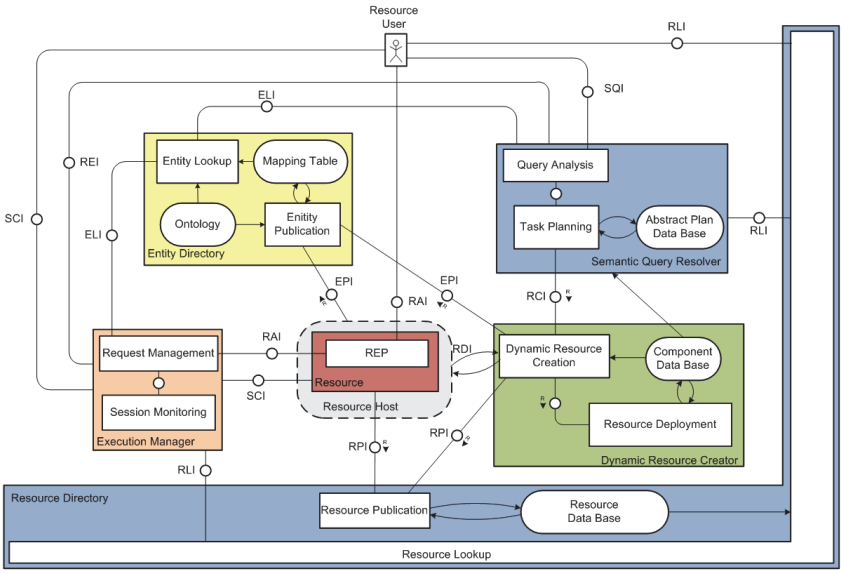

Fig. 1. RD in SENSEI architecture.

The RD contains resource descriptions of all resources available in the system at a given time. Management of descriptions stored in a RD is possible via a set of RD's interfaces which provide Resource Create, Read, Update, and Delete (CRUD) as well as Resource querying and subscribing capabilities. The role of the RD is to make the glue between Resources which advertise the operations they offer as Resource Descriptions and potential clients that look for particular functionalities. Its place in the overall SENSEI system is presented in Fig. 1.

The main components comprising an RD are the following:

- Resource Publication Interface (RPI) - responsible for publication of available resources and their descriptions.
- Resource Lookup Interface (RLI) - based on the received search parameters identifies suitable resources in the Resource Database. Subscribing interface is implemented as part of the RLI interface.
- Resource database - stores descriptions of all resources.

The RD functionality was originally implemented in JAVA programming language in combination with a MySQL database for persistent storage of data. Interaction with the resources and users was based on the RESTlet framework [3]. All requests to RD and RD responses are XML formatted messages.

Evaluation of the main technical features of the SENSEI architecture was performed in the ecoBus system [4]. The ecoBus system utilizes public transportation vehicles to carry a set of sensors across the city of Belgrade to observe a number of environmental parameters as well as events and activities in the physical world. Resources in this system are: GPS, environmental sensors on buses, bus lines, line path and estimated bus arrival time service. All resources are stored in an RD using appropriate resource descriptions. The clients can access the system using web and/or android applications which use RD's RLI interface to find appropriate resources.

III. Further RD Applications

RD component concept is one of the key components in several other projects including Smart Santander [5] and HOBNET [6]. Due to the versatility of IoT devices, i.e. resources, used in different projects, the flexibility of handling various resource descriptions has become one of the significant requirements for efficient implementation of a RD. While using XML for resource descriptions is a flexible approach, using SQL based database to store such descriptions proved to be cumbersome when new descriptions are introduced in the system as it usually means that changes of the tables and/or fields are required.

In case of the SmartSantander project, many of the resources could not be described well using the SENSEI resource description schema. Wrapping SmartSantander resources to the SENSEI defined schema for resource descriptions resulted in a rather complicated set of *Tag* elements. The following XML code is a part of the resource description of the Wisebed CTI-Testbed resource:

```
<Tag>node_name #
urn:wisebed:ctitestbed:0x1bee</Tag>
<Tag>factory_class #
de.uniluebeck.itm.tr.runtime.wsnapp.WSNDeviceAp
pFactory</Tag>
<Tag>node_type # telosb</Tag>
<Tag>node_port # XBPTDXR1</Tag>
```

It can be noted that the "key-value" data are stored in the text content of a Tag element, separated by hash symbol and hence cannot be used without prior processing. Resource wrapping degrades the presentation of XML data which is one of the key XML advantages. Further to that, some parts of the wrapped resource descriptions must use the CDATA XML's sections in order to retrieve XML validity [9]. Content of the wrapped CDATA section must be parsed using custom parsers. Resource wrapping also degrades RD's Tag querying capabilities based on the tag's text content, which is used for resource lookup as well as for resource subscription.

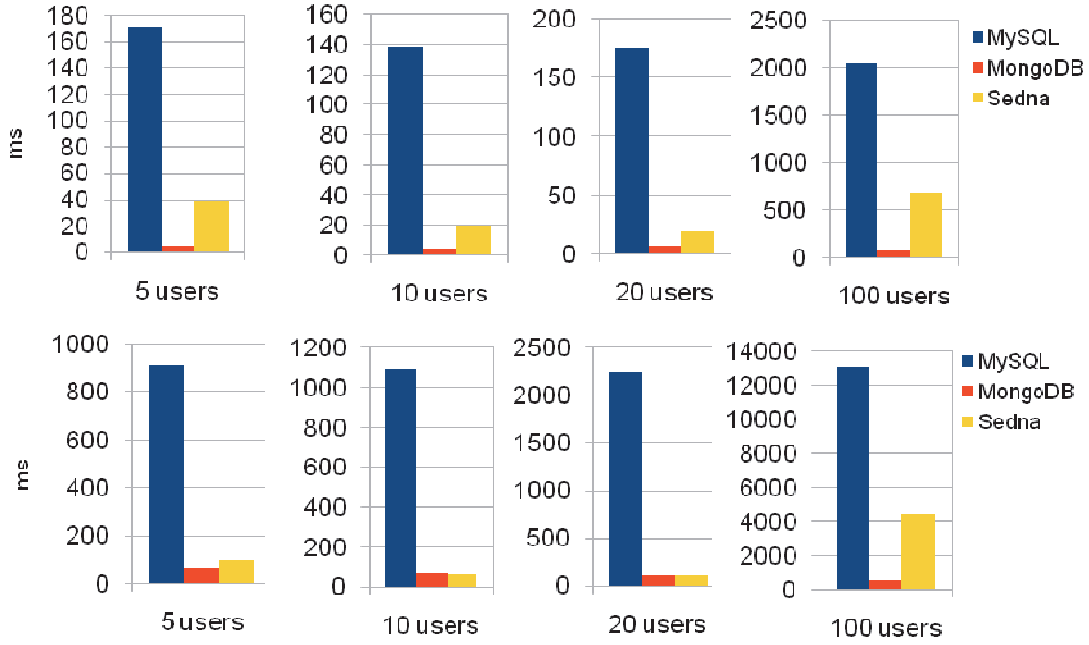

Fig. 2. Response time comparing for various numbers of simultaneous requests and 1000 and 100.000 resource descriptions in the RD respectively.

The HOBNET project adopted a combination of the CoRE [7] working group specifications for resource descriptions [8] (for constrained devices) and XML based descriptions for complex services.

Providing subscribing functionality is one of the important RD features. This procedure involves describing resources using wrapped tags and submitting them to the RD's subscribing interface. The subscribers receive notifications from the RD about the changes regarding the subscribed set of resources. These notifications also contain wrapped resource descriptions. This requires implementation of the custom, hybrid XML and text parsers on the subscribers' side thus adding complexity. Relational and XML based databases use different data types, and XML's ability to change structure in the middle of a document does not work well with the rigid table structures of relational databases. Generating XML document from a data stored in relational databases could lead to expensive JOIN operations between many tables. Also, when an XML document is stored in a relational table, information can be lost, such as element ordering and the distinction between attributes and elements.

To address the issues above and simplify the problem of storing versatile XML files, we implemented two versions of RD: the first based on a native XML database (SEDNA [10]) and the second based on a document oriented database (MongoDB [11]).

IV. Sedna XML RD Implementation

Sedna is an open-source native XML database system being developed by the MODIS team at the Institute for System Programming of the Russian Academy of Sciences [10]. Sedna is developed from scratch in C/C++ and implements W3C XQuery language and its data model exploiting techniques developed specially for this language. XQuery is a query and functional programming language that is designed to query collections of XML data [11].

The Sedna based RD is implemented in Java programming language as a web application. RESTLET libraries [13] are used for the REST web services implementation. Connection to the Sedna database is maintained using Sedna Java driver API library. Data are separated to the appropriate database collections. Collections can be considered as tables in the relational database terms. The collections store and manage XML files. The resource descriptions are stored in the collection named "rdcoll". Subscriptions are stored in the "subscriptions" collection, resources matching the subscriptions in the "matched" collection and notifications for subscribers are stored in the "notifications" collection.

The XQuery is used in the querying procedure. This enables more powerful lookup capabilities compared to the tag based lookup used in the original RD implementation. The old tag based lookup capability is kept as well as it's extended to support XQuery client lookups as well as mixing the tag and XQuery lookups.

Resource publishing, reading, updating and deleting is performed using Sedna Java driver API applied to the appropriate collection in the database. In this implementation, resource descriptions are addressed using provided Resource-ID element. If the Resource-ID is omitted during publishing procedure, a Sedna generated ID is added to the resource description. Resource reading from the native XML database does not require XML serialization from several tables, because RD's database already has an appropriate formatted resource description. All this simplifies resource managing procedures because there is no need to manage several tables during resource description storing, reading, updating and deleting. Access to resources is provided through the appropriate URL in the following format RD_URL/rd/{Resource-ID}.

The subscription interface is separated from the RLI. A subscriber can use both XQuery as well as tag based queries. The original subscription capability is extended to enable subscribers to read notifications using the HTTP's GET method. This approach is useful in the situations when subscribers couldn't provide inbound HTTP connections. Notifications can be automatically deleted after reading, if an appropriate URL parameter is used at the access time.

V. MongoDB RD Implementation

MongoDB is an open-source high-performance scalable document store written in C++. It is based on the model of a collection of documents. MongoDB uses BSON (Binary JSON - a lightweight traversable binary format) as the data storage and network transfer format for documents [12]. Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

MongoDB supports replication and high availability as well as horizontal scalability up to a thousand of machines. Dynamic (ad hoc) querying is supported in the form of BSON document query patterns. Documents can be indexed on any field to speed up query processing. Additionally, map-reduce style of aggregation processing is also available.

MongoDB supports GridFS specification for storing large binary files in the database. Range operations over binary files are possible which is especially beneficial for large files, such as videos.

Interactive work with MongoDB is enabled through the mongoDB console. The query and document syntax is JSON based. Appropriate mapping between XML resource descriptions and a JSON resource description is done. An example of a JSON resource description as rendered in the MongoDB console is given below:

```
{
   "_id" : ObjectId("4f62ec2544ae496db3b245aa"),
   "Name" : "Bus Location Sensor",
   "Resource-ID" : "urn:sensei:ericsson.com:
                    EnvironmentalSensors:busgps:3574670",
   "Access-Policy" : null,
   "Expiration-Time" : ISODate("2012-03-18T07:24:14Z"),
   "Tags" : ["Bus", "GPS", "Sensor", "Tracker", "Pančevo",
                  "14", "357467030477996" ],
   "RAI-Descriptions" : [ {
      "RAI-ID" : "",
      "Description" : "GET returns sensor values (RDF)",
      "REP-Locators" : [
         {
           "REP-Locator-Value" :
            "http://www.bustracker.rs/rephandler/gps/3574670",
           "Expiration-Time" : null,
           "Proxy-Setting" : null
         }
      ]
   } ],
   "Storage-ID" : NumberLong(20)
}
```

The MongoDB based RD is implemented in Java programming language. The public RESTful API is preserved which makes replacement of the current MySQL based implementation with the MongoDB one simple and straightforward.

As is the case with MySQL and SEDNA implementations, accessing the single resource description is done using the HTTP GET method to the URL of the following format: RD_URL/rd/{Resource-ID}. Deleting is done using the HTTP DELETE method at the same URL. Querying is enabled via the RLI interface. Connection to a MongoDB is implemented using a native Java driver.

The resource descriptions in the MongoDB RD are kept in an appropriate document collection called "rdcoll". The current MySQL based implementation assigns a unique auto-incremented identifier, called "Storage-ID", for each resource description stored in RD. MongoDB does not support auto-incrementing fields. These identifiers are exposed through REST API as a parameter of access and delete operations. Because we wanted to retain full compatibility on the API level, we created the additional document collection that keeps track of the last Storage-ID value for the purpose of implementing the sequential auto-incremented Storage-ID of published resource descriptions.

VI. Performance Evaluation

Evaluation of the response times for the main RD features is presented in this section. The response times are collected using the Apache JMeter tool [12]. Apache JMeter is an open source Java desktop application designed to load test functional behavior and measure system performance. The JMeter provides a rich set of the components including the HTTP GET, POST, PUT, DELETE methods as well as a thread manager for multi user testing and a number of helper components like timers, counters, asserts, etc. The components can be executed simultaneously using a number of threads. Complex test scenarios can be created in the JMeter tool using XML formatted test plan files.

The first set of tests is performed for a variable number of resource descriptions in the databases and a variable number of simultaneous requests. The tests are performed for 1000 and 100.000 resource descriptions and 5, 10, 20 and 100 simultaneous requests. Resource descriptions are the same, with the exception of the Resource-ID. The test results are shown in Fig. 2.

The operation executed during the RD testing was a default resource description listing. This operation returns a set of matching resource descriptions limited to the length of 10. It can be noticed that document oriented implementations have significantly shorter response times in all test conditions. In the MongoDB and Sedna based RD implementations, operations of resource listing and limiting the number of retrieved resource description are implemented using an appropriate API for collection management. The main reasons for the longer response times in the case of the relational based RD implementation are primarily due to the need to execute expensive JOIN operations between several tables. It can be noted that MongoDB, on average, performs better than SEDNA despite the fact that XML parsing and serialization is avoided in the case of SEDNA.

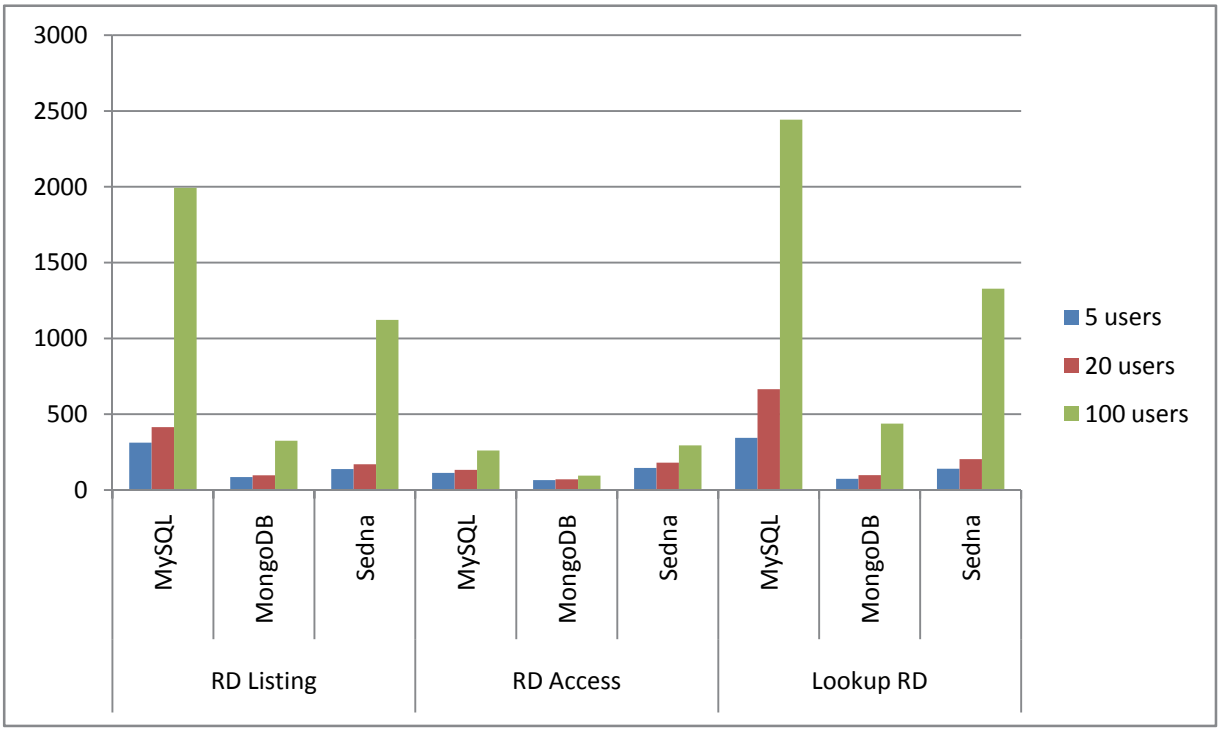

Fig. 3. Response time comparing for EcoBus RD data.

The second set of tests is performed using the real RD data from the EcoBus system [ref]. The tests are performed for the most frequently used operations in the EcoBus system: resource listing, lookup and access to the particular resource description. Resource descriptions from the EcoBus system are copied to the MongoDB and SEDNA RDs which are hosted on the same server. The test results are shown in Fig. 3.

The lookup was performed for the tag *Bus,* with 70 matching resource descriptions in the RDs. It can be noticed that the MongoDB based RD is the best performing solution in all conditions. The SEDNA RD response times are better than responses of the MySQL RD, except in the case of accessing a particular resource description. In that case the relational based RD has a slightly shorter execution time. The most frequently executed RD operation in the EcoBus system is lookup, regarding to finding REP interfaces of buses, bus-lines etc.

VII. Conclusions

The variety and heterogeneity of Internet of Things devices make it difficult to store their descriptions in relational databases as addition of new types of devices often requires changes in the database structure. In this paper we presented two implementations of a Resource Directory based on the document oriented databases which make a good fit for the structural nature of resource descriptions. The first implementation was based on the native XML database SEDNA, as the primary format used in current RD implementation is XML. The second RD implementation was based on the MongoDB. Although, it is not a native XML, it is known for its good performance, scalability and replication support which are important features in IoT systems.

Testing results show that both document oriented RD implementations perform better than the relational counterpart. This can be contributed to the usage of the expensive JOIN operation in the case of a relational database.

An additional benefit of using a document oriented database is a less rigid schema model allowing for adaptation of the document structure without imposing too much of additional burden on software developers.

Overall, the best performer in this research is the MongoDB despite the fact that XML resource descriptions were used in communication to the RD. The MongoDB is also horizontally scalable to a thousand of machines which makes it really attractive choice in environments where heavy load is envisioned.

On the other hand, SEDNA's native usage of XML, i.e. ability to directly process XML data without having to do XML serialization/deserialization first, results in increased flexibility in resource management. However, if additional representation formats must be supported, the XML parsing cannot be avoided and in that case native XML databases lose their advantage.

One of the drawbacks of the SEDNA's Java driver was the inability to set a limit on the query result set response. During evaluation, a lot of resource descriptions were similar and queries could match more than 800.000 descriptions which could lead to database deadlock due to resource exhaustion. Xquery does support limiting the response set but this ability does not solve the problem because the internal calculation still works with all matched resource descriptions.

The 32-bit version of MongoDB is limited to 2GB of database size which may be an issue in some setups. However, the 64-bit version does not have this constraint.

Further tests are planed in the context of HOBNET project and adopted CORE link format. This diversity of data formats raises the need for refactorization of the current RESTful interfaces to support different interchange formats. Using different resource description's representation formats diminishes the benefit of native XML databases.

REFERENCES

[1] V. Tsiatsis, A. Gluhak, T. Bauge, F. Montagut, J. Bernat, M. Bauer, C. Villalonga, P. Barnaghi, and S. Krco, *Real World Internet Architecture. In: Towards the Future Internet - Emerging Trends from European Research*, IOS Press, Amsterdam, April 2010.

[2] SENSEI, "*Integrating the Physical with the Digital World of the Network of the Future*", FP7 project, www.sensei-project.eu.

[3] RESTLET, "*Open source REST framework for the Java*", www.restlet.org.

[4] S. Krco, J. Vuckovic, and S. Jokic, "ecoBus – Mobile Environment Monitoring," *Towards a Service-Based Internet Third European Conference, ServiceWave 2010*, Ghent, Belgium, December 13-15, 2010, pp. 189-191.

[5] Smart Santander project, http://www.smartsantander.eu/.

[6] Hobnet project, http://www.hobnet-project.eu/.

[7] Constrained RESTful Environments (CoRE) http://datatracker.ietf.org/wg/core/.

[8] CoRE Link Format http://tools.ietf.org/html/draft-ietf-core-link-format-11.

[9] Extensible Markup Language (XML) www.w3.org/XML/.

[10] Modis group, Sedna XML DBMS, http://modis.ispras.ru/Development/sedna.htm.

[11] W3C XML Query (XQuery), http://www.w3.org/XML/Query.

[12] Apache JMeter, http://jakarta.apache.org/jmeter/.

[13] Restlet - RESTful web framework for Java, www.restlet.org/.

[14] MongoDB, http://www.mongodb.org/.

[15] Binary JSON, http://bsonspec.org/.