MasterBroker: REST oriented service broker

Conference Paper · July 2014

DOI: 10.1109/INES.2014.6909374

CITATION
1

READS
139

5 authors, including:

Dijana Kosmajac
Dalhousie University
11 PUBLICATIONS **23** CITATIONS
SEE PROFILE

Vladimir Vujovic
University of East Sarajevo
54 PUBLICATIONS **1,073** CITATIONS
SEE PROFILE

Nikola Davidović
University of East Sarajevo
21 PUBLICATIONS **171** CITATIONS
SEE PROFILE

Branko Perisic
Singidunum University
86 PUBLICATIONS **656** CITATIONS
SEE PROFILE

MasterBroker: REST oriented Service Broker

Dijana Kosmajac*, Vladimir Vujović*, Mirjana Maksimović*, Nikola Davidović*, Branko Perišić**
*Faculty of Electrical Engineering, University of East Sarajevo, East Sarajevo, Bosnia and Herzegovina
**Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
dijana.sagit@gmail.com, vladimir_vujovich@yahoo.com, mirjana.maksimovic@etf.unssa.rs.ba,
nikola.davidovic@etf.unssa.rs.ba, perisic@uns.ac.rs

***Abstract* — Considering historical development and expansion of World Wide Web, it is obvious that there is no unique tendency in terms of supporting software technologies. Moreover, vast amount of different technologies has emerged because the system as a whole became quite complex and diverse. Heterogeneity, which is characteristic of distributed systems based on Web, is one of the main challenges which system designer may encounter. The paper presents and discusses a software solution proposal which mainly relies on widely deployed Internet stack protocols and REST (Representational State Transfer) style of Web services. Further, the paper compares standardized approach based on SOAP (Simple Object Access Protocol) to a proposed one based on REST. As a result, the implementation represents a control component with broker architecture.**

I. INTRODUCTION

Despite prevalent computer networks are in fourth decade of utilization, the concept of distributed services relying on network infrastructure in complex information systems is quite recent. The reasons for such a situation can be found in relatively low level of fault tolerance and automatic recovery, availability and satisfying feedback, as well as security level in considering global service accessibility.

Development of Internet and World Wide Web in mid-1990s has enabled distributed systems to broaden beyond traditional fields, like industrial automation, military defense and telecommunications to almost all actual domains of application including electronic commerce, financial services, healthcare, government and entertainment. Different types of networks, single or combined, present infrastructure which supports distributed services architecture.

Distributed systems incorporate significant part of technological achievements which have emerged in a past few years, thus comprehension of modern information technologies is essential to their effective application.

Domain of distributed systems has great significance considering web technologies development trends, as well as architecture pattern approach. A similar research can be found in the paper [1]. The proposal is given which suggests reduction of complexity of service lookup by client request. Integration of different QoS features is essential for web service technology success. Because of increasing popularity of the technology and potential dynamic selection and integration of services, different service providers offer functionally similar services. QoS is essential factor for distinguishing such services. Main problem is the lack of mechanism which tracks QoS features during service selection. Another paper [2] is focused on early stages of software architecture development process, while system partitioning is being done on high level. The paper discusses system partitioning with reference to functional requirements and predefined quality attributes. Particularly, architectural patterns are discussed as a potential solution for balancing between these two types of requirements.

MasterBroker solution, presented in this paper, represents the foundation for the development of an open system model that can be easily optimized and upgraded. Interoperability among two or more broker systems is directly supported in a specified model. Different broker systems can communicate and interact if they follow mutually agreed message exchange protocol. This protocol is implemented by bridge components, which have responsibility of translating of broker specific protocols to mutual protocols and vice versa. Interaction between clients and services is based on dynamic model, which generally means that services can also behave as clients in the system.

This dynamic interaction model is somewhat different than traditional client server-model, where the roles are static. Considering the implementation, clients can be viewed as applications, and servers as libraries, although other implementations are also available. It is important to note that clients do not have to "know" the location of requested service. This is important feature because it allows adding new services and/or migration of existing ones to other locations while system is running.

In this paper a one possible solution of MasterBroker services, based on RESTful principle, is described. The implementation is based on set of popular web technologies, including Spring core and Spring security framework, as well as Hibernate. Further, the solution based on broker architecture offers unified view of registered distributed web services, in the manner that registered client may uses different services through single interface.

The rest of this paper is structured as follows. Section II describes development of distributed system with trends and challenges, while in section III web services are described. Section IV discusses main concept of MasterBroker service component and its architecture. An approach to security implementation is presented in Section V. Section VI concludes the paper and explains the roadmap for evaluation and other issues for future researches.

978-1-4799-4615-0/14/$31.00 ©2014 IEEE

II. Distributed System Development

Considering the theoretical aspects, there is no unique definition of a distributed system. In [3] distributed system is defined as a collection of independent computers which, from the user point of view, appears as a unified and coherent system. On the other hand, in [4] it is defined as a system whose hardware and software components are located on networked computers which communicate and coordinate their actions only via message exchange. These definitions cover wide spectrum of systems in which networked computers interact in order to execute particular task, appearing to the end users as monolithic systems.

Primary motivation for distributed system construction and utilization arises from the need to share resources [3]. The term *resource* refers to an abstract concept. Nevertheless, it most suitably depicts range of elements which can be usefully shared in a networked computer system. The term includes hardware components like disks or printers, and software defined entities like files, databases, or data objects of different type.

The second definition mentioned earlier raises a set of issues, with concurrency, lack of global clock and independent faults as the most significant ones [4].

Further, characteristics and trends of distributed systems are described, as well as challenges which system designer may encounter.

A. Trends

Based on analysis of the problem domain, the collection of major trends can be identified in the area of distributed system development. The most significant are:

1) Network technologies and modern Internet – Internet, identified as set of interconnected computer networks of different types (WiFi, WiMAX, 3G, Bluetooth), is gradually transitioning to ubiquitous internetwoking. In addition, Internet can be considered as a global distributed system which enables users, no matter what is their location, utilization of services, like World Wide Web, e-mail or data transmision. The set of services is extensible, i.e. it can be expanded by adding hardware and/or software elements.

2) Mobile technologies and computing – Technological advances in device minimization and wireless networking have led to integration of small and portable devices to distributed systems. Ubiquitous computing [3], along with mobile computing are representatives of these technologies.

3) Distributed multimedia systems – Multimedia support can be defined as support feature for a certain range of media type, as integrated component in system [4]. Such system is expected to support data storage, transmission and presentation of discrete media type, including text messages or images, as well as continual media type, including audio and video records.

4) Distributed systems as services – Along with expansion of distributed system infrastructure, a certain number of enterprises started to promote distributed resources as services. The resources are offered by corresponding service providers. In a past few years, cloud computing has emerged. The cloud is defined as set of Internet based applications, storage and computer services created to satisfy arbitrary user needs.

B. Challenges

During the design phase of distributed systems, system designer must be aware of the issues that might potentially arise. Consequently, certain, yet significant challenges can be identified. As the scope and scale of distributed systems and applications is extended the same and other challenges are likely to be encountered. Set of major challenges during design and development includes: component heterogeneity, openness, security, scalability, fault handling component concurrency, transparency and quality of service [4].

III. Web Services

In a real business world, in terms of distributed systems, flexibility of Internet technologies is a significant factor [5]. Processes and systems are becoming more and more complex. In early stages of development, distributed systems were decoupled, individual systems. On the other hand, nowadays the tendency of their coupling into one global distributed system – Internet emerges.

Traditional methods of scalability and distribution handling are not applicable to new ideas and technologies. Centralization, which was formerly precondition for harmonization and system control, now seems to be inadequate solution concerning system scaling. Because of that, the decentralization tendency is introduced which supports the cooperation of heterogeneous systems.

Service oriented architecture (abbrev. SOA) is a technology which can offer a solution for traditional problems related to distributed system design and generally represents flexible set of design principles used during process of system development.

A. SOAP

SOAP (Simple Object Access Protocol) [6] is a technology which is diverse enough to allow using different transport protocols. Stack of standards use HTTP as transport protocol, but other protocols can be used, such as JMS and SMTP. Whereas SOAP model tunnels well in request/response model, it can easily tunnel through existing firewalls and proxies, without modification of SOAP protocol itself, and generally can use existing architecture. On the other hand, due to possible redundancy of XML format, SOAP can be far slower than competitive middleware technologies (like CORBA). In case of small messages, this is not the problem. If HTTP is used as transport protocol without WS-addressing or ESB (Enterprise Service Bus), roles of sides are fixed. Only one side can use service of another.

B. REST

REST (Representational State Transfer) [7] by itself is not an architecture, but it represents a set of constraints which, when applied during system design, build up software architectural style. If all REST guidelines are implemented, as pointed out in [7], the resulting system will have specific roles for: data, components, hyperlinks,

communication protocols and information consumers. In the same paper, the author suggests REST after evaluating significant number of network resources and technologies available for construction of distributed applications. Without any principles and constraints, resulting applications would be hard to maintain and extend.

C. *Documenting Web Services*

World Wide Web for a certain amount of time did not have formal resources which could be used for web application documentation. Absence of web app interface formal description introduced difficulties in web component development which do not have direct communication with browser (web services). The two specifications have emerged, and every specification has an aim to provide resources which computers can process for formal REST style web application interface description: WADL (Web Application Description Language) [8] and WSDL 2.0 (Web Services Description Language) [9] HTTP binding extension. WSDL is a language which can be used for web service definition, but if WADL is considered as an option, WADL is lightweight, easier to understand and easier to write than WSDL. In [10] a brief comparative analysis between two specifications is given, in terms of RESTful services.

Despite programming patterns, based on technologies like Java and .NET, are relatively easy to understand and implement, there is a possibility of automatic code generation using service metadata. A static service description can be used for presenting available addresses and resource representation format. Advantage of WADL usage is due to support for automation and code generation based on description or vice versa. To illustrate how WADL can be used, language authors created WADL2Java [11] tool, which enables creation of clients in Java, minimizing manual code writing used for interaction with service described in WADL.

WADL can be useful as description language for CRUD services, and in addition for automatic code generation. As client and service cooperate through resource lifecycle, URI and representation format, it is irrelevant how client app was written (generation or manual). WADL descriptions can facilitate client side maintenance when changes occur on service side.

IV. Design of Access Control Component

MasterBroker component, as a sample implementation broker architectural pattern, is a web based app for access and usage one or more services through one component. Being web based, and being dependent on technologies used in implementation, it introduces the limitations upon communication paradigms and system infrastructure.

A. *MasterBroker Architecture*

The system, RESTful by its nature, has a structure of broker architectural pattern [12] which involves decoupled components interacting via remote service invocation. Referring to base definition of the broker, it is responsible for coordinating communication, including request forwarding, as well as result and exception transfer. Because of such features, broker architecture is found suitable for the implementation.

Central component which implements brokerage among set of services and clients is the MasterBroker Service, as shown in Fig. 1. The component provides published interface towards CustomServices, which can vary in number (1 to n). The interface enables third party services to register on MasterBroker Service, and consequently, publish their resources through unique interface managed

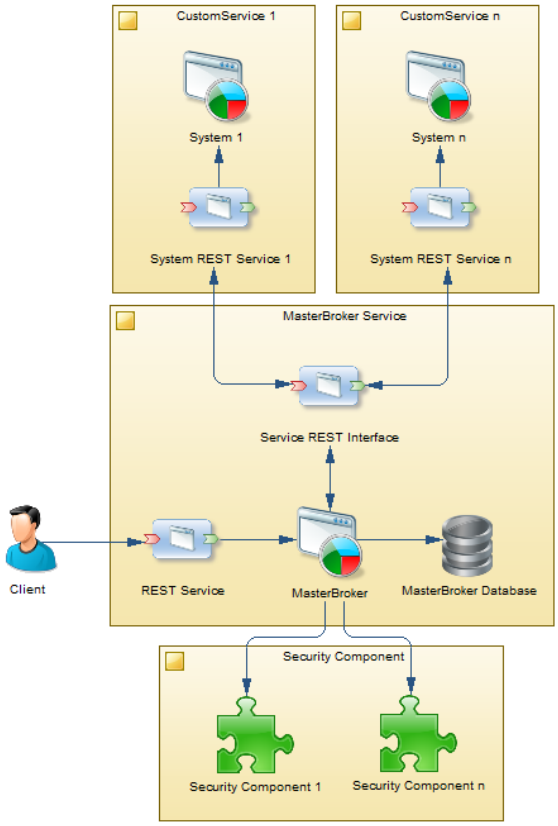

Figure 1. System architecture model

by the system. The system has an internal repository which holds all metadata about registered users (clients and services). In addition, whole operating logic of the system relies on data stored in the repository. Part of repository contains a structure for persisting service WADL definition. Based on this part, application has ability to:

- persist service WADL document during service update or registration;
- generate WADL definition for client based on client access rights.

Next, MasterBroker Security Component is implemented as a component decoupled from the system, as shown in Fig. 1. Besides that, the MasterBroker has an interface towards Security Component implementations, so the actual system may have different behaviors in the terms of security. Security implementation has responsibility of managing authentication and authorization processes, which implies existence of repository on the actual Security Component side.

Protocols, used in intercommunication among elements (Fig. 1), primarily rely on Internet stack protocols integrated within exchanging messages. WADL definitions mentioned earlier are crucial part of metadata being exchanged. The Custom Services send their definitions during registration process in order to allow access to remote clients through MasterBroker corresponding interface. Later, WADL definition generated by MasterBroker is sent to client. That definition differs from original Custom Services' definitions, as it is generated using access policy defined in Security Component. An illustration of exchanging protocols is represented in Fig. 2.

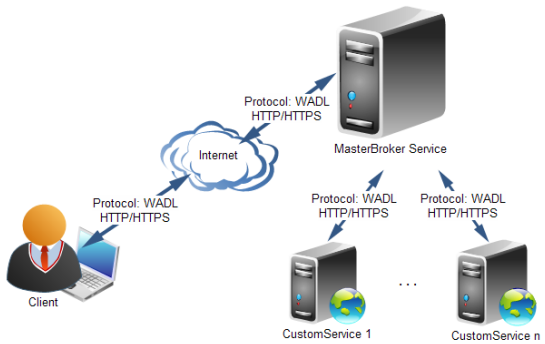

Figure 2. Protocols among components of the system

Using proposed architecture of distributed system, whole communication is transferred to MasterBroker, and Client does not need to know the locations of CustomServices, but only location of the MasterBroker. Migration and security of CustomServices are also handled by MasterBroker.

B. *Registering a Custom service to MasterBroker*

Fig. 3 depicts process of registering a service to MasterBroker system. The process includes sending

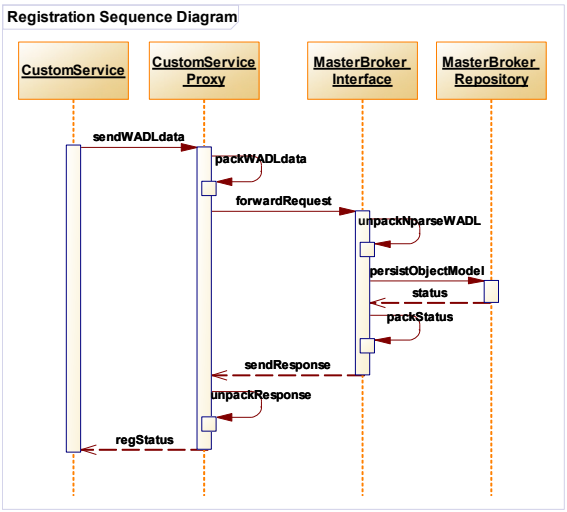

Figure 3. Registration sequence diagram

WADL description of the service, in order to allow that system obtains "knowledge" about published service resources.

- CustomService application is in runtime. Via exposed MasterBroker interface, CustomService sends WADL and registration specific data to the system.
- CustomService proxy packs all data, including WADL into message suitable for transfer and forwards it to MasterBroker service.
- MasterBroker service unpacks message and parse WADL description to object structure. Data objects are persisted to MasterBroker repository.
- After persisting object data, repository returns status of persistence (failure or success) to MasterBroker service.
- MasterBroker packs results into suitable message and forwards it to CustomService proxy.
- CustomService proxy receives response, unpacks data and returns to CustomService unpacked status.

C. *Interaction among Elementsin MasterBroker*

Sequence diagram, presented in Fig. 4. illustrates situation when client sends request to CustomServices over the MasterBroker service. The diagram shows a

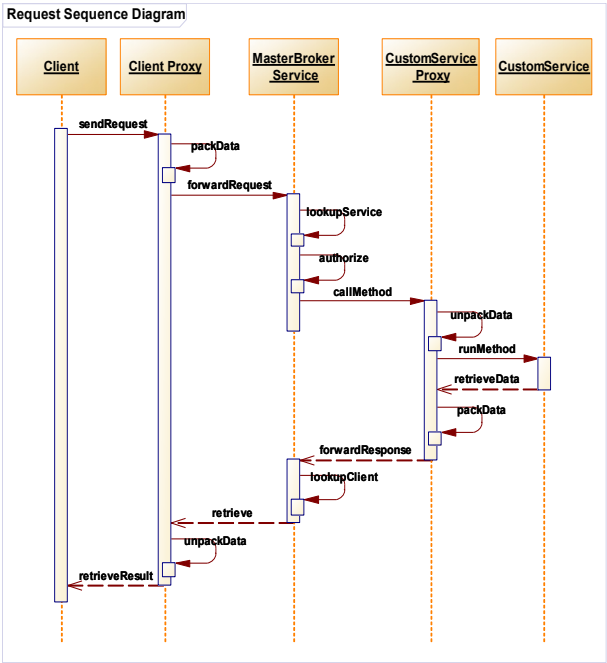

Figure 4. Request sequence diagram

synchronized request that forces the client to be blocked until receiving the service response (nature of HTTP imposes this behavior).

- Client application is in runtime. During the runtime (client is already logged, depending on Security component of MasterBroker service) client calls resource method of CustomService.
- Client proxy packs all parameters and other important data into message and forwards it to MasterBroker service.
- MasterBroker service lookups requested resource URI in its database using parameters in request. Based on URI found, it forwards message corresponding CustomService proxy.
- CustomService side proxy unpacks all parameters and other data including, for instance, method

type. CustomService side proxy calls corresponding resource.
- After resource method execution, CustomService returns result to CustomService proxy, which packs result or exception and forwards packed message to MasterBroker service.
- MasterBroker service forwards response to client proxy.
- Client proxy receives response, unpacks data and returns to client unpacked response. Client process continues with fetched remote service data.

D. Implementation

MasterBroker, as a system that enables the access to services collection, is implemented as web based application, and, in terms of communication, relies on Internet stack protocols. During implementation phase Spring framework [13] was used together with the elements of Spring Security framework. The main reasons to utilize it are extensibility and customizability features. As a database, MySQL Community Edition [14] is used, because of simplicity of testing, as well as because of the fact that relevant management tools are legally and freely downloadable.

Using Spring framework forces MVC design pattern approach, where views, data models and controllers are decoupled. Concerning data access, instead of traditional solution based on stored procedures on database side, Hibernate framework [15] is chosen, so the data access logic is moved to application domain. Because of this, it was necessary to generate object oriented model which represents direct database structure mapping to object model. Instead of XML description of mapping in Hibernate, EJB [16] annotations are used, which directly describe relations among objects in relational sense.

Hibernate, as well as Spring framework, due to ample background logic, introduces certain latency during execution of requests sent to application. Communication latency between service and broker is in used test case is slight and can be neglected, as applications are deployed in same servlet container. Namely, database batch insert is used through Hibernate, which can result in memory leakage when big data structures are sent. As described in [17], there are certain recommended code and configuration optimizations, so the queries have relatively small execution time.

One of mechanisms used for optimization is a so-called lazy object initialization i.e. referencing lists (in relational modem one-to-more). In the example 1, configuration of an object is shown for lazy list fetching. For relations more-to-one eager type is recommended.

Example 1. Referencing objects data retrieving configuration

```
    public class Resources implements Serializable
    {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Basic(optional = false)
        @Column(name = "RESS_ID")
        private Integer ressId;
        …
        @OneToMany(mappedBy = "ressId", fetch = FetchType.LAZY)
        private List<Documentation> documentationList;
        @OneToMany(mappedBy = "ressId", fetch = FetchType.LAZY)
        private List<Resource> resourceList;
        @JoinColumn(name="REGSRV_ID",referencedColumnName=
"REGSRV_ID")
        @ManyToOne(fetch=FetchType.EAGER,cascade=CascadeType.ALL)
        private Service regsrvId;
        …
```

If large collections are sent to database, which, in this application, is possible situation (sending WADL definition, parsing), it is recommended to define size of batch and to order queries insert and update, because Hibernate by default configuration executes query by query. `Hibernate.config` file in the example 2. shows one configuration:

Example 2. Hibernate configuration

```
    <hibernate-configuration>
        <session-factory name="sessionFactory">
          …
          <property
name="hibernate.order_inserts">true</property>
          <property
name="hibernate.order_updates">true</property>
          …
        </session-factory>
    </hibernate-configuration>
```

V. Distributed Web Services Security

The need for security of information integrity and privacy and other resources belonging to individuals and organizations, is present in physical, as well as in digital world. Distributed systems consist of concurrent processes which access to distributed resources via message exchange in a network environment. The network environment can be insecure and can contain untrusted components which leads to term distributed system security in network.

Distributed system security issues can be split into certain number of elementary issues. In web oriented technologies domain there are standard solutions for the elementary security issues. They are identified as follows: authentication, authorization, data integrity, confidentiality, availability and privacy.

A. Security Mechanisms

Secured system design in a real world is based on balancing between implementation cost and security requirements span. Namely, collection of techniques which can be implemented in best scenario for securing of processes and inter-process communication can be strong enough to defend from almost any kind of attack, but implementation and usage of it, on the other hand, can cause huge costs. Nevertheless, badly specified security measures can influence on possibility of executing substantial actions guaranteed to authorized users.

B. Custom Token Authentication

Custom token authentication [18] is most simple method for implementation. The process consists of two steps: first, unique token generation per API registered user; second, every registered user sends generated token for authentication with every request to service. As token is being send with every request, this method is not secure, because fraud can copy valid token and use it without authorization. There is no way to determine if such request is valid or not.

Characteristic of HTTP authentication [18] is that username and password are sent in plain text base64 in header HTTP Authorization. Username and password must be sent with every HTTP request.

Relatively simple way of securing client-service communication has serious flaws. As this type of authentication uses base64 code, username and password can be easily read. A solution for this is presented and described in the paper [19].

VI. CONCLUSION

Distributed systems are quite complex in a real world in terms of their physical characteristics; system scale, heterogeneity level and defined security level. Rising the level of abstraction in the process of design introduces modeling as a mechanism for dealing with modern systems complexity.

The paper presents current and future trends of distributed systems development. In addition, it briefly discusses some of actual technologies that are widely used and general concepts and constraints which a system designer should be aware of. As an illustration of discussed concept the architecture model is proposed which, on the software level, has aim to incorporate and gather background components including communication paradigms and elements with reference to roles of every element. Architecture pattern depicted – broker, provides more complex design with use of key elements including client-server model and solutions based on distributed components and distributed web services.

A control web component with broker architecture is presented which generally implements the following features:

- *Location transparency*, which is achieved through masking of remote service real location from clients and vice versa;
- *Changeability and extensibility*, which is achieved with feature of registration of theoretically unlimited number of clients and services on control component;
- *Portability and interoperability*, which is achieved through hiding of operating system details and network system on which broker relies, from clients as well as services with support of Internet communication protocols;
- *Reusability*, which is achieved through utilization of existing services during client application implementation.

Besides achieved features, certain disadvantages are identified:

- *Limited efficiency*, as utilization of intermediary component in communication contributes to request-response process latency. In the project there are certain optimizations inside of system, but latency still exists in augmented form compared to simple client-server architecture.
- *Lower fault tolerance*, as requests towards broker are realized to execute synchronous, i.e., wait for remote service response. In this way, there is possibility for faults on client applications. Further, unavailability of some registered services at the specified moment disables proper execution of all dependent client applications. As a solution for this [4], in literature is often mentioned replication of critical components.

There are several reasons for choosing plain technology REST over well standardized SOAP. As mentioned earlier, SOAP has very robust exchange messages, which can, at some point, carry a lot of redundant data (due to XML messages). On the other hand, REST offers freedom to designer to choose other representations, like JSON, for message exchange. Next, REST seems to win battle in current trends of ubiquitous and mobile technologies, where leading companies, like Google, force using REST through their products. Lastly, SOAP has well standardized security framework, while REST offers arbitrary technologies.

REFERENCES

[1] T. Rajendran, P. Balasubramanie, R. Cherian, „An Efficient WS-QoS Broker Based Architecture for Web Services Selection," International Journal of Computer Applications, t. 1, br. 9, pp. 110-115, 2010.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] N. Harrison, P. Avgeriou, „Pattern-Driven Architectural Partitioning: Balancing Functional and Non-functional Requirements," Digital Telecommunications, San Hose, CA, 2007.

[3] A. S. Tanenbaum, M. Van Steen, Distributed systems: Principles and paradigms (2nd edition), New York: Pearson Prentice Hall, 2007.

[4] G. Coulouris, J. Dollimore, T. Kindberg and G. Blair, Distributed Systems, Concepts and Design (5th edition), Boston: Addison-Wesley, 2012.

[5] N. M. Josuttis, SOA in Practice, Sebastopol: O'Reilly Media, 2007.

[6] S. Graham, D. Davis, S. Simeonov, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. Koenig, C. Zentner, Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI (2nd edition), Indianapolis: Sams Publishing, 2005.

[7] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Irvine: University of California, 2000.

[8] M. Hadley, „Web Application Description Language," W3C, 31 August 2009. [online]. Available: http://www.w3.org/Submission/wadl/. [Accessed 12 December 2013].

[9] D. Booth, C. K. Liu, „Web Services Description Language (WSDL) Version 2.0 Part 0: Primer," W3C, 26 Jun 2007. [online]. Available: http://www.w3.org/TR/wsdl20-primer/. [Accessed 10 December 2013].

[10] T. Takase, S. Makino, S. Kawanaka, K. Ueno, C. Ferris, A. Ryman, „Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0," Tokyo Research Laboratory, IBM Research, Tokyo, 2008.

[11] Oracle, „Wadl2Java," Oracle, 2013. [online]. Available: http://wadl.java.net/wadl2java.html. [Accessed 10 December 2013].

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sornmerlad, M. Stal, A System of Patterns (1st edition), West Sussex, England: John'Wiley & Sons, 1996.

[13] Spring Framework, „Spring Projects," Spring, 2014. [online]. Available: http://projects.spring.io/spring-framework/. [Accessed 10 January 2014].

[14] MySQL, „MySQL Community Edition," MySQL, 2013. [online]. Available: http://dev.mysql.com/downloads/mysql/. [Accessed 10 December 2013].

[15] Hibernate, „Hibernate ORM," Red Hat, 2013. [online]. Available: http://hibernate.org/orm/. [Accessed 10 December 2013].

[16] Oracle, „Enterprise JavaBeans Technology," Oracle, 2013. [online]. Available: http://www.oracle.com/technetwork/java/javaee/ejb/. [Accessed 10 December 2013].

[17] C. Bauer, G. King, Hibernate in Action, Greenwich, USA: Manning Publications Co., 2005.

[18] J. Sandoval, RESTful Java Web Services, Mumbai: PACKT Publishing, 2009.

[19] D. Kosmajac, V. Vujović, "Information systems security and security extension in Jersey RESTful framework", TELFOR 2012, pp. 1556-1559, ISBN 978-1-4673-2983-5, DOI 10.1109/TELFOR.2012.6419518, 2012