Composite Web Services for E-Activities

STOICA CRISTINA, STOICA VALENTIN, IONESCU FELICIA, CIOBANU COSMIN
Faculty of Electronics, Telecommunications and Information Technology
University Politehnica of Bucharest
1-3 Iuliu Maniu Bvd, Postal Code 061071, Sector 6, Bucharest
ROMANIA
cristina@tech.pub.ro, vstoica@yahoo.com, http://info.tech.pub.ro

Abstract: This paper presents a solution for design, implementation and testing an e-commerce application based on advanced distributed technologies and tools. The current demands for enterprise applications require the possibility of running on heterogeneous systems (the platform independence), a minim of functional dependencies between software components of the applications and a standardized manner for integrating the applications and presenting the functionalities. The most recent response is provided by SOA (Service Oriented Architecture) and Web Services technology. But the complexity is done by composing the functionalities in a specific order and simply enough to permit easy future changes. In this area, WS-BPEL (Business Process Execution Language for Web Services) is important for composing Web Services in business processes. Thus, we illustrate creating composite Web service for distributed application over Internet.

Key-Words: - service oriented architecture, distributed technologies, composite Web services

1 Introduction

E-business applications became today more and more complex as data volume and functionalities increases. Hereby, the need to split these in independent modules that can be easily and efficiently managed concern software developers.

Most e-activities are client-server applications that imply the existence of a server (database server or Web server or any application server) and clients. This can be a thick client, which means that at this level is implemented also the business logic of the application. Or it can be a thin client, that do not do anything else than display the data received from the server. In this case, the business logic of the application is implemented on server-side.

Because the actual applications must deal with a huge data volume and complex business logic, three-tier architecture is most appropriate and frequently used. The client application represents the user interface, the data are handled by a database server and data processing is implemented on another level (an application server or a Web server) that also deal with client's requests.

At database design level, software developers must consider aspects concerning concurrency control and fast data access [1]. Also must apply techniques to improve the communication [2].

The client applications do not communicate directly with database server when request data. At logic business level, the data is processed using distributed objects. The main disadvantage of using distributed objects is the fact that the objects have state and lifecycle and a client is limited to use them only during this period of time. Web services represent a stateless technology based on a new architectural style, SOA (Service Oriented Architecture), used to obtain a maximum functional independence between the different software components of a complex application [3]. This also leads to an efficient development and maintenance.

2 SOA and Web Services

A SOA contains three roles: a *service requestor*, a *service provider*, a *service registry* and three operations: *publish*, *find* and *bind*.

A service provider creates a *service description*, publishes it in one or more service registries and receives messages invoking the service from one or more service requestors. Thus, a service provider represents the server-side inside the client/server relation between the service requestor and the service provider. A service requestor finds certain service description published in one or more service registries and uses it to bind or invoke provider's services. Thus, a service requestor represents the client-side inside the client/server relation between the service requestor and the service provider.

A service registry presents the service descriptions published by the service providers and allow searching these descriptions by the service

requestors. Its role is matching the service requestor and the service provider.

After that, the service registry became unnecessary, the interaction evolve between the service requestor and the service provider directly. The client sends a SOAP (Simple Object Access Protocol) request message to the server, that generates based on client request a SOAP response message [3]. The SOAP is XML (Extensible Markup Language) based and provide a matching mechanism between SOAP messages and HTTP (Hypertext Transport Protocol) that is the most used communication protocol over Internet.

In distributed systems and service oriented architectures area, Web services has gain ground in the last years. The Web services infrastructure providers like IBM, Microsoft, W3C and Sun have published their own Web services definitions [4]. A conclusion of all those definitions can be as follows: a Web service is platform and implementation independent software and it can be described using a description language like WSDL (Web Services Description Language), published in a service registry, detected through a standard mechanism like UDDI (Universal Description Discovery and Integration) [3]. Web services can be invoked over a network through an API like JAX-RPC (Java API for XML) based RPC (Remote Procedure Call) [5].

A very important Web services characteristic is the composition [6]. Web services can be composed with other services using languages like WS-BPEL (Business Process Execution Language for Web Services) [7]. A BPEL process specifies the strict order for invoking the participant Web services. The invocation can be sequential or parallel. With BPEL a conditional behavior is possible. For example, invoking a service can be dependent from a value of a previous invocation. Cycles can be build, variables can be declared, copied or initialized, etc. Thus, complex processes can be defined based on algorithms. Each BPEL process integrates more activities: *invoke* for invoking methods of other services, *receive* for waiting an invocation from a client, *reply* for generating answers, *assign* for data manipulation block, *switch*, etc. A BPEL process defines links with the partners, using a *partnerLink* tag. For a client, a BPEL process looks like a simple Web service. A BPEL process is defined as a composition of the existing Web services. The interface of the new composite Web service uses a set of *portType*-s providing operations like any other service. BPEL processes receive invocations from clients. One of them is the BPEL process user that makes the initial invocation. The others clients are Web services invoked by BPEL processes and call a callback method returning the results. A schematic BPEL process is shown in Fig. 1:

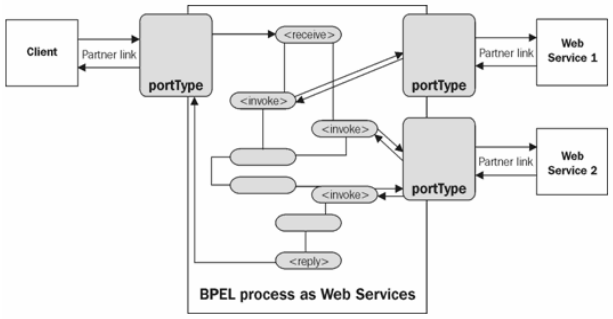

Fig.1 Sample of BPEL process

3 The trip-planning application

To get to the final software product, that is the trip planning application, we follow the steps: analyzing the application demands, application modules design, choosing the technologies and modules implementation, installation and testing the application.

3.1 The application demands analysis

We suppose the following scenario: a person wants to plan a trip. He must select the city of departure, arrival city, the departure time, the return date, the number of accompanying persons, and the quality of desirable services. First, he makes an on-line ticket reservation. Then, he makes reservation for a hotel room in the arrival city. He also can rent in that city a car for a specific period.

Thus, we need to keep information about flights, hotels, cars, and clients and to offer services to access this information when a client wants planning a trip, to make his options, to pay for requested services.

3.2 The modules design

Because we want an application based on Web services, we need for our scenario a Web service for flight ticket reservation, a Web service for hotel room reservation, a Web services for car rental, a Web service for the payment, and a Web service for payment and invoicing. Each one of these Web services must have an associated database. Thus, we need a database with information about flights, a database with information about hotels, a database with information about cars and a database with information about client's cards.

For ticket reservation, the Web services display all the flights available at the requested time, price

and duration of each one. The client chooses one of the displayed flights, and then fills the ticket reservation form. After the reservation will be invoked the Web service for payment and invoicing. If bank transaction *commits*, this service will send an invoice to the client. If transaction *abort* because there isn't any card for the client, or there isn't enough money on respective card then the service will delete previous reservations. If there aren't flights at requested time or the client do not chose any flight then the client is interrogated for deleting the other reservation (hotel and/or car). The same algorithm applies to the next steps, which are reserving the hotel room and car rental.

Thereby, to achieve such functionality, the Web service for flight reservation must offer the following methods: displaying a list of airports linked by flights, returning a list of flights available at the requested time with available places, modifying in the database the number of occupied places corresponding to selected flight, registering personal client data in the database, issuing an invoice, deleting the reservation and the places when the client cannot reserve hotel rooms or car, deleting issued invoices in the previous circumstances ant returning the money back to the client account.

The Web services for reserving hotel rooms and car rental are similar.

The Web service for payment and invoicing is a composite Web service. For reserving a flight this service can issue an invoice invoking a method of the reserving flights Web service or can delete the reservation invoking appropriate method of the same Web service. For payment, the composite Web service invokes a method of the payment Web services.

Fig. 2 shows the Flights database diagram:

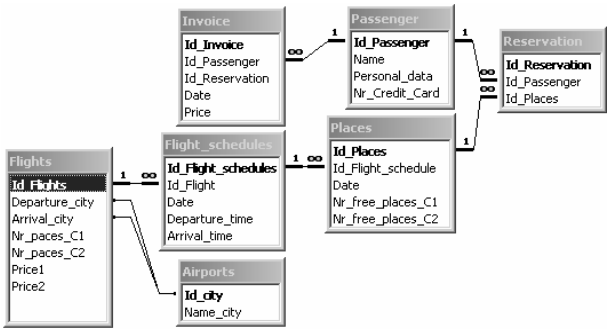

Fig. 2 Flights database diagram

Fig. 3 shows the UML (Universal Modeling Language) activities diagram for flight ticket reservation [8]:

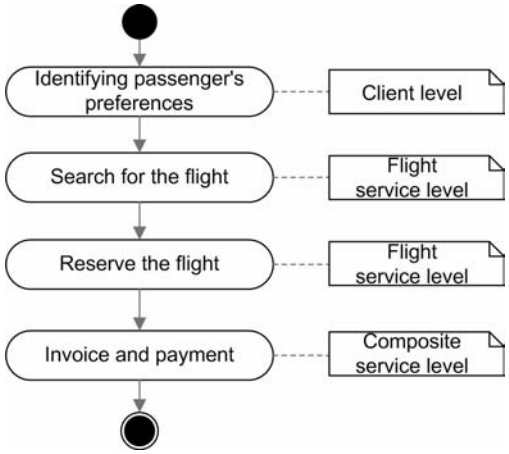

Fig. 3 The UML activities diagram for flight ticket reservation

Fig. 4 shows the composite Web service for payment and invoicing for hotel services:

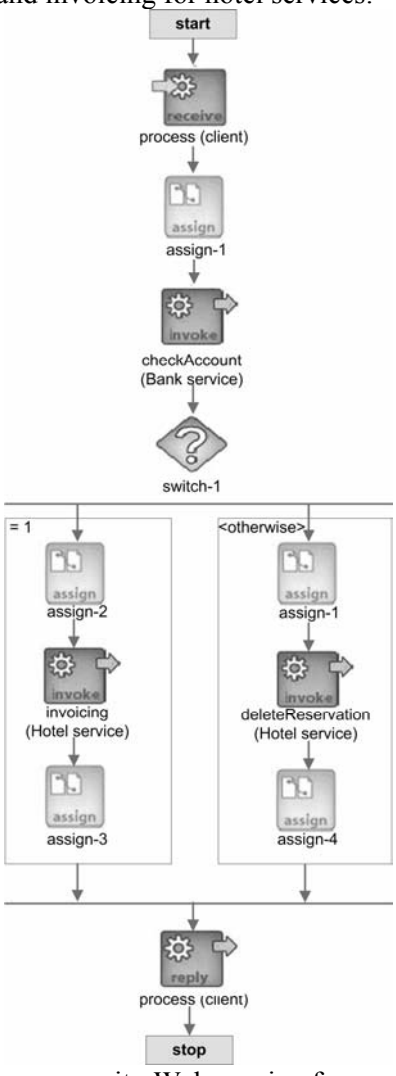

Fig. 4 The composite Web service for payment and invoicing for hotel services

Fig. 5 shows the UML sequence diagram of Web services for flight ticket reservation:

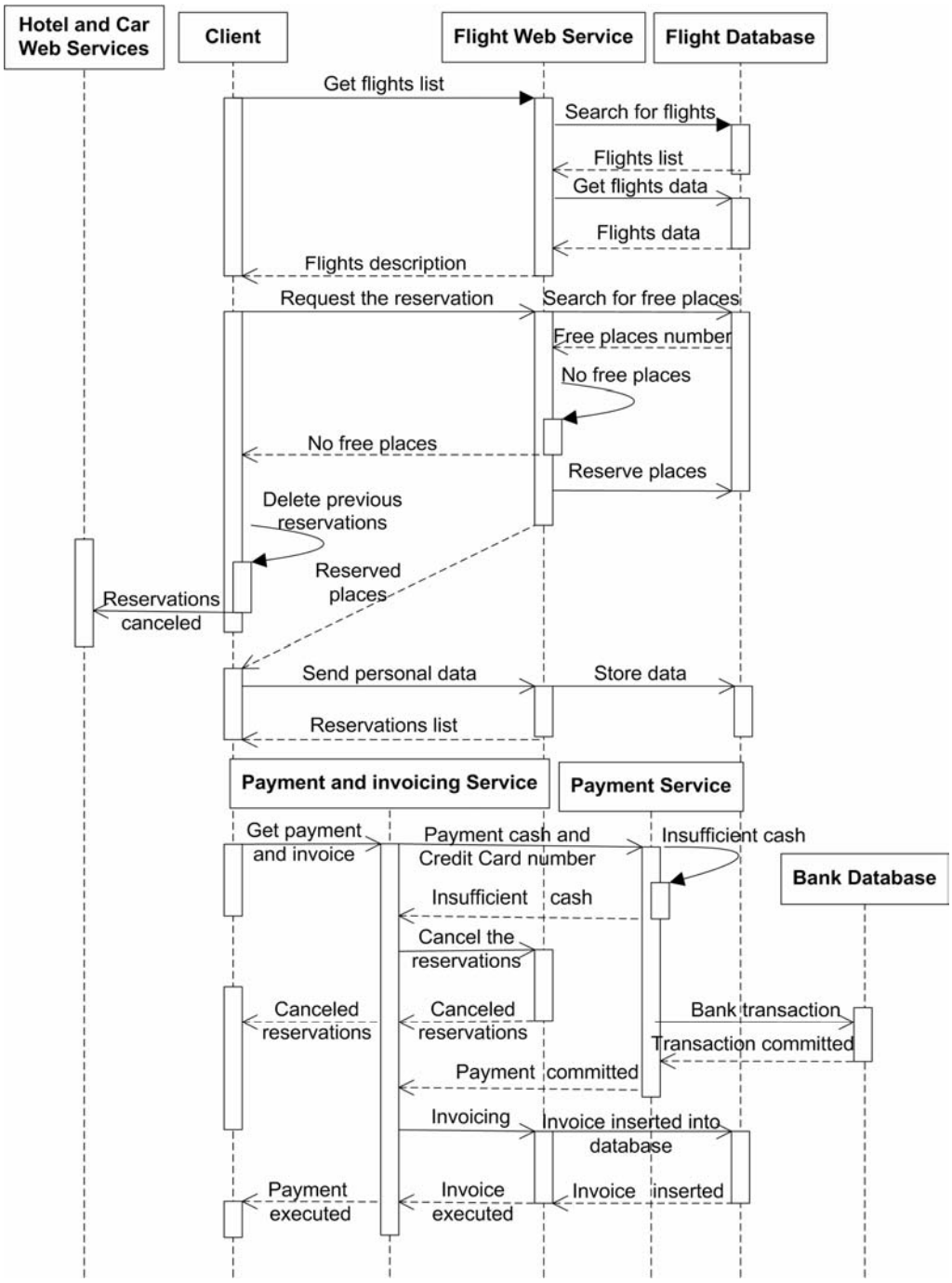

Fig. 5 The UML sequence diagram of Web services for flight ticket reservation

3.3 The modules implementation

For implementing the data support level, that means the databases, we chose MySQL 4.1 [9].

For implementing the application business logic level, we chose the J2EE (Java 2 Enterprise Edition) technology, that offers an interface for database accessing, JDBC (Java Database Connectivity) and also a support for creating and developing simple Web services and Oracle BPEL Process Manager for development and execution of composite Web services, that means BPEL processes [7].

For implementing Web clients who access the Web services, we use the JSP (Java Server Pages) technology [10]. A JSP page can change dynamically its content depending on the client's actions.

3.4 Components deployment and testing

Fig 6. shows the UML diagram of deployment components:

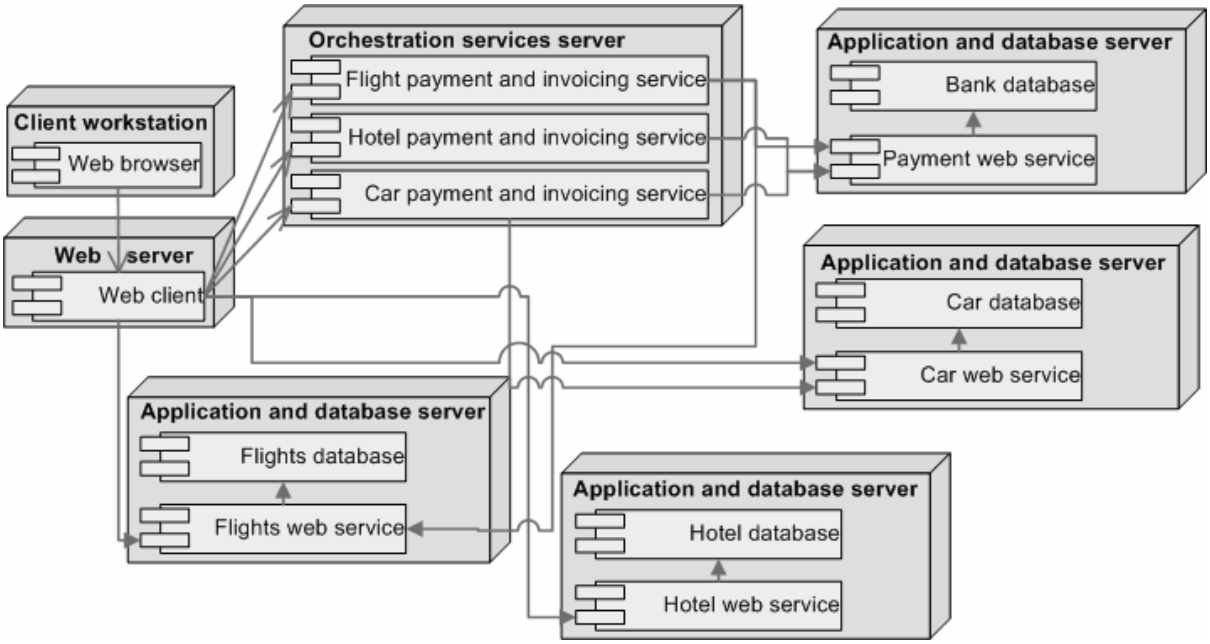

Fig. 6 The UML diagram of deployment components

We measure the response time of Web services to client's requests. The response time is defined as time between the start of a client request and the end of the response received by the client's terminal. The tools used for measure the response time was Mindreef SOAPscope 4.1 [11]. This tool allows the capture of SOAP messages and the measurements of different Web services parameters.

In the first test was invoked the search_Id_city(String name) method from the ticket reservation Web service running ten times with the same argument. Fig. 7 shows the result:

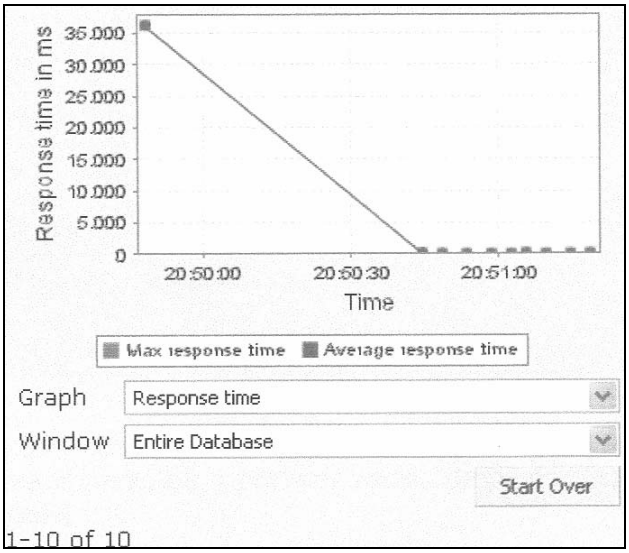

Fig. 7 The response time for a repetitive method invocation

At the first invocation the response time is higher that the response time of subsequent invocations because a servlet is created and then reused.

The second test measured the response time for flight ticket payment and invoicing composite service. Fig. 8 shows the result:

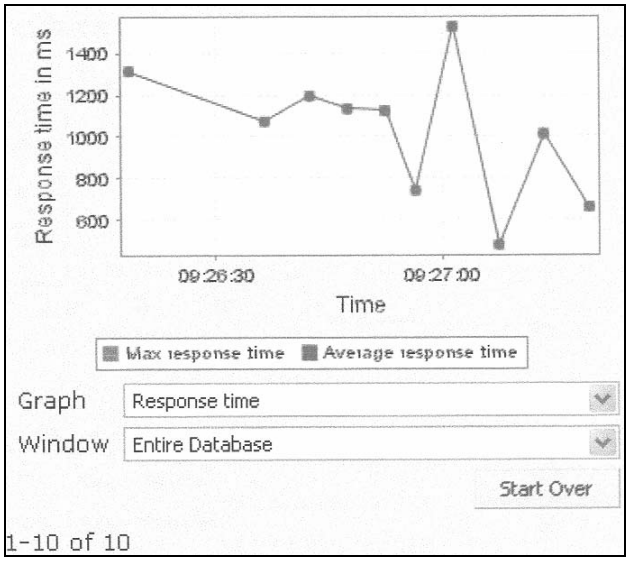

Fig. 8 The response time for the composite service

To compare the response time of composite service with the response time of the simple services we measure independently the response time for checkAccount() method of banking service and invoicing() method of flight ticket reservation service. Both methods are invoked by the above composite service and measured in the Fig. 8. We

maintain the same input parameters. The next two figures show the result:

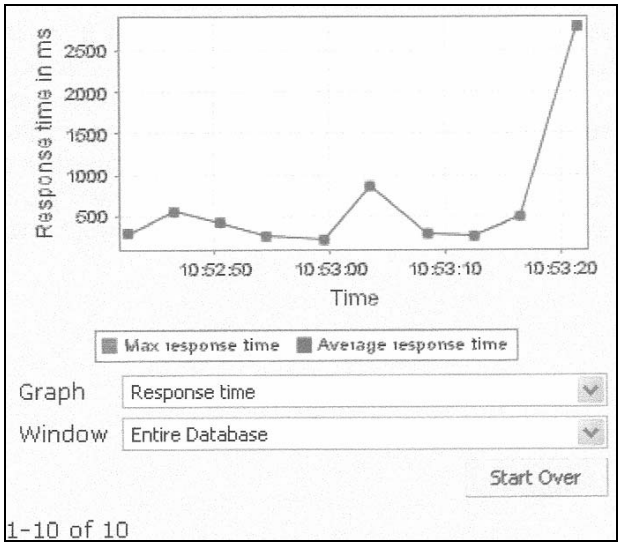

Fig. 9 a) The response time for checkAcount() method

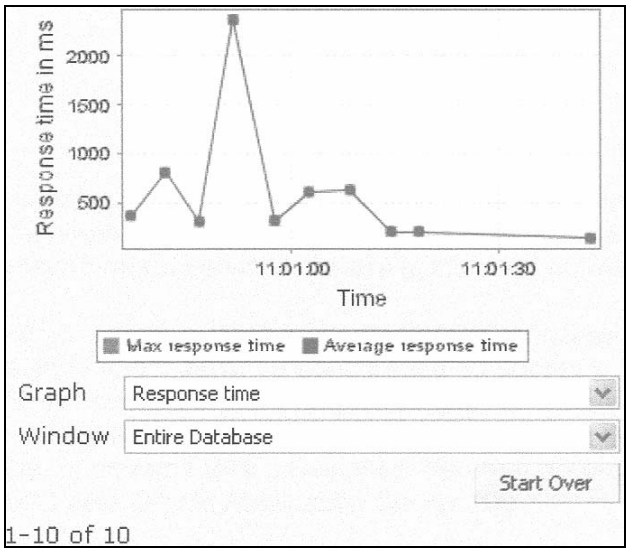

Fig. 9 b) The response time for invoicing() method

Adding the response times for separately invocation of the two methods, it can be observed that the response time for the composite service is 10%-20% higher that the cumulative response time for the two separate services.

4 Conclusion

Although the utilization of composite services is a slower solution, it is accepted due to the huge advantage of implementing complex processes reusing Web services that are already implemented and can be easy discovered searching in services registry.

Acknowledgements

The Romanian National University Research Council under grant EL100608 supports this work.

References:

[1] Stoica Cristina, Stoica Valentin, Ionescu Felicia, Design Aspects for Fast and Concurrent Data Access, *WSEAS Transaction on Information Science & Applications*, Issue 5, Volume 1, November 2004, Rethymno, Creta, Grece, 24-26 October 2004, pp. 1283-1288, ISSN 1790-0832.

[2] Felicia Ionescu, Cristina Elena Stoica**,** George Valentin Stoica, Şerban Balamaci, Advanced Replication Techniques for E-activities, *Proceedings of International Conference on Computational Intelligence for Modeling, Control & Automation CIMCA 2005 jointly with International Conference on Intelligent Agents, Web Technologies & Internet Commerce IAWTIC 2005*, Vienna, Austria, 28-30 November 2005, pp. 904-909, IEEE Computer Society, Volume II, ISBN-10: 0-7695-2504-0.

[3] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo Neyama, *Servicii Web cu Java™-XML, SOAP,WSDL si UDDI*, TEORA Bucharest, 2003.

[4] http://www.sun.com/software/sunone/faq.xml

[5] Eric Armstrong, Jennifer Ball, Stephanie Bodoff, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock, *The J2EE™ 1.4 Tutorial*, Sun Microsystems, 2004.

[6] Nikola Milanovic, *Current Solutions for Web Services Composition*, IEEE Internet Computing, 2004

[7] http://otn.oracle.com/bpel, *BPEL – Learn by Example.pdf*

[8] Karin Palmkvist, Bran Selic, *Advanced UML Modeling*, 2000

[9] http://www.mysql.com/

[10] Hans Bergsten, *JavaServer Pages*, O'Reilly, 2002

[11]http://www.mindreef.com/products/soapscope/ss_3_4_upgrade.php