Krassimir Markov, Vitalii Velychko,
Lius Fernando de Mingo Lopez, Juan Casellanos
(editors)

New Trends in Information Technologies

I T H E A
SOFIA
2010

Krassimir Markov, Vitalii Velychko, Lius Fernando de Mingo Lopez, Juan Casellanos (ed.)

New Trends in Information Technologies

ITHEA®

Sofia, Bulgaria, 2010

ISBN 978-954-16-0044-9

First edition

Recommended for publication by The Scientific Concil of the Institute of Information Theories and Applications FOI ITHEA

This book maintains articles on actual problems of research and application of information technologies, especially the new approaches, models, algorithms and methods of membrane computing and transition P systems; decision support systems; discrete mathematics; problems of the interdisciplinary knowledge domain including informatics, computer science, control theory, and IT applications; information security; disaster risk assessment, based on heterogeneous information (from satellites and in-situ data, and modelling data); timely and reliable detection, estimation, and forecast of risk factors and, on this basis, on timely elimination of the causes of abnormal situations before failures and other undesirable consequences occur; models of mind, cognizers; computer virtual reality; virtual laboratories for computer-aided design; open social info-educational platforms; multimedia digital libraries and digital collections representing the European cultural and historical heritage; recognition of the similarities in architectures and power profiles of different types of arrays, adaptation of methods developed for one on others and component sharing when several arrays are embedded in the same system and mutually operated.

It is represented that book articles will be interesting for experts in the field of information technologies as well as for practical users.

General Sponsor: Consortium FOI Bulgaria (www.foibg.com).

Printed in Bulgaria

ISBN 978-954-16-0044-9

C\o Jusautor, Sofia, 2010

SOA PROTOCOL WITH MULTIRESULTING

Michał Plewka, Roman Podraza

***Abstract**: This paper presents a framework for distributed results in SOA environment. These distributed results are partially produced by a service and passed to a client program on demand. This approach is a novelty in SOA technology. Motivations for this new design is figured out and a sketch of implementation is outlined. A configuration for the new tool is suggested and finally its efficiency is compared to a conventional web service implementation.*

***Keywords**: SOA, multiresulting, client-server, web services.*

***ACM Classification Keywords**: D.2.11 Software Architectures*

Introduction

By multiresulting we understand a case when a remote service is returning a collection of data. Usually the client program has to wait until the service is completed to obtain the complete result. In a number of applications the server processes a huge amount of data what makes the waiting time very long. However in many cases it would be possible to start processing if the client program received only part of results returned by a remote method. So it will be desirable if the remote service could return collection of data in number of steps. For example an attempt to migrate a knowledge discovering system to Service Oriented Architecture focused our attention on the issue of dealing with processing of large collections of data in a distributed environment. A client program usually organized a processing as a sequence of calls to remote services and in many cases the services could be used in a pipeline style if the data could be processed in a continuous or quasi-continuous manner. So we decided to it create such a framework which will allow to return partial results on demand of the consumer of remote services. We named these partial results as multiresults (and the whole approach as multiresulting) to underline their usefulness when they are available separately. Currently remote services are obtainable mainly with technology of web services. They are easy to develop, to maintain, and what is very important, easy to reuse on many different machines thanks to SOAP (Simple Object Access Protocol). Our framework proposal should not be inferior to the known and accepted SOA concepts, but it should provide a satisfactory remedy to the presented drawbacks of existing approach.

Architecture

The prepared mechanism has to work asynchronously and should return data in such a way which allows to split it to many parts. The platform should be used as an independent component which works in a transparent way. The solution obviously has to contain two main layers - the first one is the server side and the second is concerned with the client layer. How the communication between these layers should work? Protocol HTTP was chosen, due to its simplicity. This kind of protocol is often associated with a servlet technology [B. Basham, 2004].

A client sends information about full qualified class and its method name which should be initiated on the server side. Arguments are sent in serialized form as an array of various elements. Then on the server side the request is being intercepted by the specified servlet which invokes method on an Enterprise Java Bean (EJB) [P. Debu, 2007]. In fact, the EJB deserializes arguments and invokes method in specified class using the reflection mechanism. Then a unique identifier is returned to the client. In next iterations the client layer must use this identifier to retrieve the data from the remote method. In the mean time the remote method is being executed on

the server and the results are being collected by a queue. When the client asks the server for a part of data available results are being retrieved from the appropriate queue recognized with the help of the identifier which was generated in the first client-server interaction. Finally, this identifier should be used to identify the correct process. When the remote method is completed then the EJB is notified that the service has been accomplished no more partial results will be returned. The rest of results in the queue has to be returned to the client and with the last portion a special flag is being set to indicate that no more active pooling is required from the client side. It is very important to set a time interval for client interactions. If this value is too small the system performance may be degraded by an inefficient network traffic. Moreover, the time interval for the client-server interactions influences how often the remote method delivers the partial results. If this interval is too short it should be adjusted to a higher value to get more data per interaction.

Implementation

This paragraph presents a framework, which was designed and implemented to support multiresulting as a new way of client-server communication. Some procedures how it should be used are proposed and explained.

A developer should prepare a remote method (or service) firstly. The class which contains it must extend class *MultiResultProcess*, which is provided in the framework. The remote methods producing multipart results have to be public, void and annotated with predefined annotation *@MultiResultMethod*. The annotation has a single parameter *maxElementsPerInteraction.* It defines a maximum number of collection elements which could be returned in a single interaction.

When a partial result is ready to be passed to the client a special method must be invoked by the remote service itself. This method (*newDataArrived(Serializable[])*) takes an array of serializable elements as its parameter. The method is inherited from parent class and it performs all actions required to process the available partial results. The function cannot use just the *return* statement, because it still has to produce further results. Finally, when the whole operation is finished another inherited method (*endProcess()*) should be invoked. It notifies the container controlling the remote call, that the remote method has been finished successfully. Each remote method supporting multiresulting mode of communication has to be completed in this way. If method *endProcess()* is invoked for a given remote method more than once its successive call will be futile and an appropriate exception is thrown.

When a remote method has been prepared it can be installed in Java Enterprise Application context. This could be done by creating dynamic web project and modifying *application.xml* to use the specified multiresult EJB and map the servlet to a required path. File *web.xml* should contain entries which map class *ResponseServlet* to a specified path. It could be done using tags <*servlet*> and <*servlet-mapping*>. The full enterprise application configuration is described elsewhere.

Finally when correct paths are set and libraries are imported, the client side may be prepared. To invoke a multiresult method deployed on a remote server, class *RequestProcessorFactory* has to be used. This class enables producing objects supporting remote method calls. Class *RequestProcessorFactory* has method *initiateProcessing(String), which* returns object of type *RequestProcessor* enabling method initiation. As a parameter the method gets a service name which is defined in a configuration file presented in the next paragraph. This object of *RequestProcessor* has a method called *Process(Serializable…)*, which handles the whole communication with the server side. To track information about availability of partial results produced by a multiresult remote method a special listener (*DataListener*) must be set. It has two methods, first one receives portions of the remote results and the second one is invoked when the remote method is terminated.

Server side has to be deployed in JEE Application Server environment, so if one decides to use it, the server must be also provided. The installation is trivial and does not differ from any normal application installation.

Configuration

Configuration of the framework is very simple. File *multiResultConfig.xml* should be created in the package resources. To configure a service firstly it is necessary to define the root element <*configuration*>. It contains number of <*service*> sections. Each of them contains the following tags:

- <name> - defining name of the remote service,
- <targetServletEndpoint> - specifying location of the remote servlet,
- <processor> - fully qualified name of class handling the communication between a client and the remote service working in multiresult mode; standard implementation provides class *SimpleRequestProcessor*, however some elaborated versions may be developed by extending class *RequestProcessor* and applied,
- <qualifiedClassName> - fully qualified name of class containing the remote method,
- <methodName> - name of the remote method,
- <delay> - integer number specifying minimal delay (in milliseconds) between successive interactions of the client and the remote method.

The client program can have configured many remote methods working in the multiresult mode. They can be deployed in many different locations and the client program may organize their parallel execution receiving partial results as quickly as they are ready. It is worth mentioning that by receiving partial results the client program is notified that the remote methods are in a continuous progress. This may be a very important feature in organizing distributed processing with some backup capabilities.

Efficiency

The presented framework of multiresulting was compared to classics web services. A test benchmark included searching data using some specified criteria. Amount of data to be searched was unknown as well as amount of results satisfying the criteria, so it was impossible to assess expected time of execution. This benchmark was quite suitable for applying multiresulting, because the data matching the criteria could be returned partialy to the client. The efficiency of multiresulting appeared to be very good. The overall time of benchmark execution was smaller for the multiresulting mode. It was not obvious result because more communication overhead is required in this way of processing. What is very important the first response from the remote service came just after the request and then the client could retrieved the first set of data. In the web service case a large amount of data was returned in a single block. In general for a number of tests the multiresulting mode was approximately 10% faster than the web service. Probably it was caused by using xml marshalling in web services while our multiresulting mode employs native Java serialization mechanism.

This solution was also assessed by number of users. They were expected to subjectively evaluate both approaches and express their estimation in scale from 0 to 10 points. Most of the users noticed and underlined higher speed of processing in the multiresulting mode. The speed-up was more evident for the larger result set. However the most striking was effect of immediate response from the program run in the multiresulting mode. In contrast, a long delay until the program exploiting web services responded in any way was in general unacceptable by the users. And it was the basic psychological advantage of the new proposed methodology. All users were convinced that the multiresulting mode is significantly better than classic web service, while dealing with results of large dimensions. When software developer proposes a service returning collection of data, the new option supporting the multiresulting approach should be concerned.

Applications of the Multiresulting Mode

When should we use the multiresulting framework instead of standard approach? The general answer to this question is not as simple as it may look. The first obvious answer could be as follows: “Use it when a huge collection of data is sent to client”. This is of course true, but the multiresulting mode could be used in many other situations. Look at possibility of returning Data Transfer Object to(DTO) the client. DTO is a snapshot of database or any other stateful resource. DTO object contains attributes which are set by back-end of application and when all of them are collected the object is sent to client. In the multiresulting mode the attributes could be returned partially, what could improve performance.

More interesting situation happens when a remote service uses some other remote methods itself and then their answers are put together and returned to the client. In a standard solution if one of the remote results comes very late or is not available at all, the method has to wait for it (maybe forever) to complete the full set of partial results or throw some kind of timeout exception. In this situation client cannot get any results. In this situation the multiresulting mode could be used and is real remedy for the problem. If n-th remote system will not answer, the client will retrieve at least n-1 first answers or all except the failed one. This will make client side less vulnerable to back-end malfunctions and in general any distributed processing may be more efficient and more reliable. This second feature can be achieved by repeating calls to remote services and/or their backups if they were unable to provide their results. In the multiprocessing mode the repeated requests may be much better targeted then in atomic, coarse-grained web services.

Not negligible characteristics of the multiprocessing mode is user’s feeling that whole process is responsive and progressive. So the multiresulting mode should replace the solutions when processing takes a long time. Lets imagine situation when user does some action and then waits few minutes to complete it. He may think that abnormal situation occurred and the process should be initiated once again. To avoid such unwise behaviour some kinds of progress bar are applied. But in remote processing environment such progress bars cannot present the level of advancement of the initiated process. Applying our framework it is possible to implement reasonable indicators of progress in processing. Very often some parts of final results may be presented by a client program in parallel to continuous remote processing.

On the other hand there are situations were standard solutions are better. When a small amount of data is retrieved from the remote processing there is no gain from applying the multiresulting mode. If the communication traffic is a bottleneck of the distributed system then applications of the multiresulting should be restricted. Otherwise the multiresulting using active pooling may increase the traffic and as result the response from remote services may be delayed even more than when applying web service technology.

Conclusion

Using the new kind of remote services could be a good alternative to the standard solutions. The multiresulting mode is a bit faster than web services and could return data on demand. It is configurable and implementing a client is not a hard task thanks to the support of the implemented framework.

The development time in which the multiresulting solution could be launched in real product environment is very important. It looks that this time can be short, because after implementing business methods it is only necessary to annotate them and add few entries to application descriptors. For now most of internet application are deployed in full application servers like IBM Websphere, JBoss, etc. so the multiresulting may be used easily in such environments. This kind of service is a new one, it requires much more testing in a real development process and this experience may show some more advantages or disadvantages of this solution.

Acknowledgment

The paper is published with financial support by the project ITHEA XXI of the Institute of Information Theories and Applications FOI ITHEA (www.ithea.org) and the Association of Developers and Users of Intelligent Systems ADUIS Ukraine www.aduis.com.ua.

Bibliography

[B. Basham, 2004] B. Basham, K. Sierra, B Bates. Head First Servlets & JSP. Ed. O'Reilly, 2004.

[P. Debu, 2007] P. Debu, R. Reza, L. Derek. EJB3 in action, Manning Co. 2007.

[E. Freeman, 2007] E. Freeman, E. Freeman. Head First Design Patterns, O'Reilly 2007.

[JEE] Java Enterprise Edition documentation, http://java.sun.com/javaee.

Authors' Information

Michał Plewka *– Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland; e-mail: plewka.michal@gmail.com*

Major Fields of Professional Activities: Java Developer, SOA technologies.

Roman Podraza *– Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland; e-mail: R.Podraza@ii.pw.edu.pl*

Major Fields of Scientific Research: Artificial Intelligence, Knowledge Acquisition.