IADIS International Journal on Computer Science and Information Systems
Vol. 3, No. 1, pp. 1-13
ISSN: 1646-3692

MONITORING SERVICES ON ENTERPRISE SERVICE BUS

Ilona Bluemke, Marcin Warda *Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland*

{I.Bluemke}@ii.pw.edu.pl

ABSTRACT

At the Institute of Computer Science Warsaw University of Technology a module for monitoring services in Service Oriented Architectures (SOA) was designed and implemented. The module is able to monitor services on the ESB (Enterprise Service Bus) level even, if the services are executed on different servers. The whole context of the flow is provided to the user. The architecture of this module is briefly presented. This tool was used to measure several parameters in real enterprise integrated architecture e.g.: the effectiveness and the usage of resources in some flows. These experiments are described and some conclusions are given. The monitoring module may be very useful in the maintenance of complex SOA systems.

KEWORDS

SOA, quality of service, monitoring, platform integration

1. INTRODUCTION

The definition of Service Oriented Architecture - SOA [4] is following: “An architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners.” The real power of SOA and Web services becomes apparent when various constituents are added, removed, replaced, or upgraded without adversely impacting the whole system. SOA- and Web services-based solutions allow products from different vendors, running on different platforms, to work together. Despite attempts to standardize, a typical IT infrastructure is heterogeneous and it makes the management a very complex task. The problems of management in SOA architectures is discussed in [5].

With the development of the Service Oriented Architecture (SOA), organizations are able to compose complex applications from distributed services supported by third party providers. Under this scenario, large data centers provide services to many customers by sharing available IT resources. This leads to the efficient use of resources and the reduction of operating costs. Service providers and their customers often negotiate utility based Service Level Agreements (SLAs) to determine costs and penalties based on the achieved performance levels.

Service-based approaches are widely used to integrate heterogenous systems. Web services allow for the definition of highly dynamic systems where components (services) can be discovered and quality of service (QoS) parameters negotiated at run-time. This justifies the need for monitoring service at run-time.

At the Institute of Computer Science Warsaw University of Technology a tool for monitoring services in Service Oriented Architectures (SOA) was designed and implemented. This tool was used in some experiments in real enterprise integrated architecture. These experiments are described and some conclusions are given. The monitoring module differs significantly from existing monitoring modules. Our program monitors services on the ESB (Enterprise Service Bus) level even, if the services are executed on different servers. Other programs are able to monitor business processes eg. Optimize for Process in webMethods or do not provide sufficient information if the execution is distributed on several servers. The monitoring module may be very useful in the maintenance of complex SOA systems.

The remainder of the paper is organized as follows. Section 2 describes some literature approaches. Section 3 introduces the overall system architecture and presents some implementations details. The experiments with the monitoring program are presented in Section 4. Section 5 contains some conclusions.

2. RELATED WORK

In [1] is presented how to monitor dynamic service compositions with respect to contracts expressed via assertions on services. Dynamic compositions are represented as BPEL (Business Process Execution Language) processes which can be monitored at run-time to check whether individual services comply with their contracts. Monitors can be automatically defined as additional services and linked to the service composition. Two alternative implementations of the monitoring approach: one based on late-binding and reflection and the other based on a standard assertion system are described.

The paper [6] proposes a framework for monitoring the compliance of systems composed of web-services with requirements set for them. This framework assumes systems composed of web-services that are coordinated by a service composition process expressed in BPEL4WS (Business Process Execution Language for Web Services) and uses event calculus to specify the properties to be monitored. The monitorable properties may include behavioural properties of a system which are automatically extracted from the specification of its composition process in BPEL4WS and/or assumptions that system providers can specify in terms of events extracted from this specification.

In [7] the quality model suitable for capturing and reasoning about quality aspects of multichannel information systems is presented. In particular, the model enables a clear separation of modeling aspects of services, networks, and devices. Further, it embeds rules enabling the

evaluation of end-to-end quality, which can be used to select services according to the actual quality perceived by users. In [7] a long list of references to papers concerning different aspects of quality of services can also be found.

3. MONITORING MODULE

However there are some monitoring programs (some of them mentioned in section 2), we decided to design and implement a monitoring module for integration product webMethods [10]. Existing monitoring modules are able to monitor business processes e.g. Optimize for Process in webMethods. The implemented module is able to monitor services executed on different servers on the ESB (Enterprise Service Bus) level. Other programs monitoring ESB, are dedicated to only one server or do not provide sufficient information if the execution is distributed on several servers. Our goal was to monitor ESB services executed on distributed servers and to provide full context of the flow. Such information may be very useful for administrators of integrated systems.

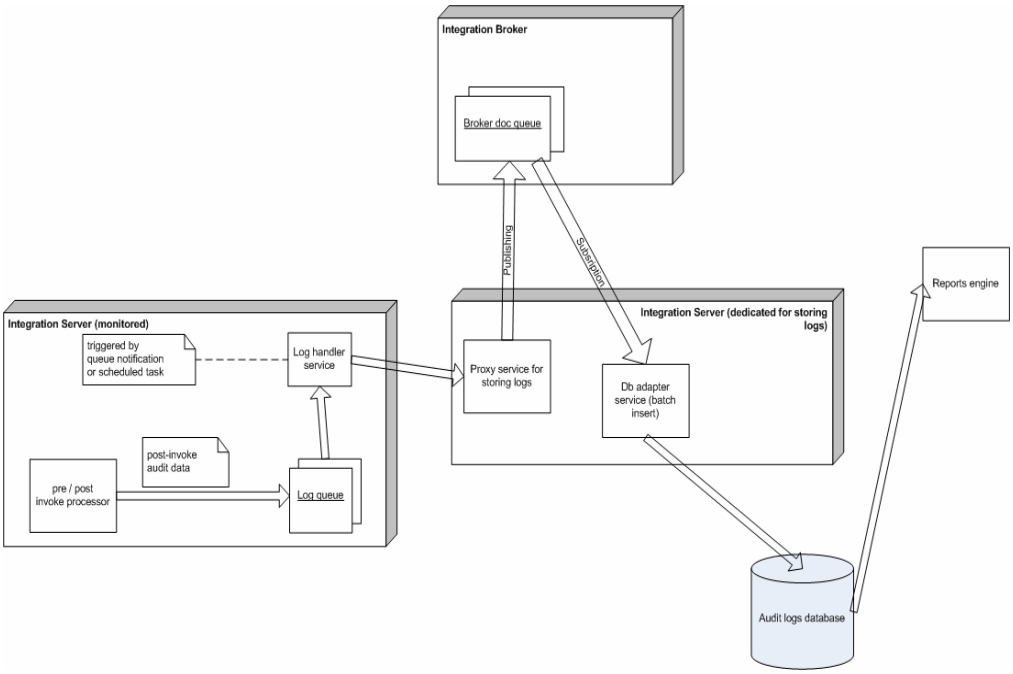

Figure 1. The architecture of monitoring module

The module for monitoring services on an integration platform contains processes responsible for:

1. collecting information concerning service calls based on a „pre/post invoke" and storing it in memory of a local server (log queue),

2. elaborating the log queue: store logs in a file on local server, write them to the data base (using connection of the local server or send them to the central service which will write logs to the data base),
3. evaluating logs in data base: statistics calculation, deleting ancient logs.

The basics ideas of a monitoring module are presented in Fig.1. The implementation details of this module are described in [9, 2].

3.1 Information collection

To collect the information concerning service calls a mechanism provided with the integration server i.e. custom „pre/post invoke" is used. Custom classes responsible for the call of web service can be added to store appropriate data in the memory of local server. The data are stored synchronically to the service call so the time to execute the service is increased by time necessary for writing data. We are convinced that this solution is more efficient than asynchronical data writes. For the asynchronicall writes a copy of objects and data must be made. Some experiments show that this process needs several milliseconds. The collected data are stored in object of the `mwr.service.performanceMonitor.beans.AuditData` class which is inserted into the log queue on the local server. To be able to decompose the execution time of a flow into the execution times of several services the whole context of the call must be stored. This context is also passed by the pre-invoke mechanism which is able to detect a service of an external server e.g. sending a message to Broker or a SOAP (Simple Object Access Protocol) call. This context contains following information:

- name of a user which started the flow,
- identifier of the flow,
- name of the service in the top level,
- identifier of previous server,
- transaction identifier.

Transaction identifiers can be used to trace the flow on different physical servers. The user name and the name of the top level service are used to set the log filters.

3.2 Information storing

On the local server a queue of objects containing data describing services is kept (fig.1). Writes to this queue can be made from different threads concurrently so they are synchronized. When the length of queue reaches a defined level a special service responsible for writing to the data base is called. This service is operating asynchronically. The queue parameters e.g. maximal length (`maxQueueSize`), the number of threads servicing logs (`maxLoggingThreads`) and others can be configured. In case of problems with the access to logs data base the queue may become very long, even use all server memory. If the maximal queue length (`maxQueueSize`) is reached (which may be caused with inability to write logs) the newest logs are covering the oldest ones, some logs will be missed but server will be operable. The monitoring module should not disturb in normal operation of servers.

Writing many logs to the data base can slow down the execution of other flows on this server. Direct writes from servers to data base need connections between server and data base. Connections with many servers on the integration platform may be a problem for the data

base. We decided to built a dedicated service on the integrated server (webService) responsible for all writes to the log data base. Logs writes are implemented in two services. The first one is sending logs to integration broker and the second one is extracting these logs and sending them to the data base. The first service manages a log queue in the integration broker and enables normal operation, even if there area some efficiency problems with the access to the data base. The service responsible for physical writes to the log data base should be installed on a dedicated server, connected with the integration broker, so writing to the log base will not obstacle normal platform operation.

4. EXPERIMENTS

The implemented monitoring module was used in several experiments in real SOA environment. The integration platform was built with 6.1.5. webMethods product [10] and contained:

- webMethods Integration Server,
- webMethods Broker,
- WmJdbcAdapter implementing JDBC protocol and enabling the connections with data bases in the integrated environment.

The environment contained five logical layers:

1. core-tier – abstract business services layer,
2. adapter-tier - contains adapting services enabling the connections between business services and physical resources of integrated systems (e.g. billing systems, client management system),
3. front-end-tier – covers adapter services provided for clients applications,
4. business-process-tier – layer providing business processes on the platform,
5. admin-tier – administrator server collects information about the availability of platform elements. This server was also used to write logs from monitoring module.

Admin server was using its own message broker. The integration environment had also configuration and operational data bases. The operational data base was also used to store logs from the monitoring module. In section 4.1 some technical details of the environment used for experiments are given. The results of experiments are presented in sections 4.2 - 4.6.

4.1 Environment for experiments

Each logical layer of integration servers had two instances on the server. Servers were working on 32-bit computers IBM Blade Server dual-core with 4GB RAM operation memory and operating system RedHat Enterprise Linux v.3. Integration Broker was located on Sun-Fire-V440 computer working under SunOS 5.9 operating system. Data base (engine Oracle 9i) was deployed at the same computer. Integrated systems were also using Oracle data bases (versions 8 or higher).

Calls of services from client were made with SOAP-RPC protocol, API webMethods implemented in Java or HTTP (GET) protocol used for direct transfers of xml files. Resources of integrated systems were accessible trough PL/SQL statements. In communication the JDBC adapter was used. The SOAP-RPC protocol was also used in synchronic internal calls between servers. For asynchronical calls, the message broker was used.

The experiments were made in an real environment of a mobile phone operator. The integrated platform consisted mainly of systems for clients and clients demands managements. The platform has approximately 300 services and 20 business processes. The analysis were made for selected, following flows:

1. `x.core.serivce:getServiceParamsValues` – getting parameters for client' services (synchronic)
2. `x.core.serivce:getAvailServices` – getting services available for the client (synchronic)
3. `x.core.service:getActiveServices` – getting active services for the client (synchronic)
4. `x.core.om.service:serviceModificationOrder` – putting an order for the modification for client' service (asynchronic)
5. `x.core.infoservice:infoservice` – checking resources available for the client in her/his electronic wallet (synchronic)
6. `x.core.contract:getContractData` – getting data from the client' contract (synchronic).

The logs were collected during one week of functional testing of the integrated platform. The functional testing was caused by new code release for the integrated systems. During testing the load of the system was significantly lower than normally (only 5% of the normal, working load). The flows during testing were not evenly distributed and this phenomena can be seen in the results of experiments. It was not possible, due to the owner constraints, to monitor normally working platform.

In next sections some analysis of the stored by monitoring module logs are shown. The results present possible application of the monitoring module. In a short paper not all application domains may be presented, more analysis are given in [9]. The performance analysis should not be made during the functional testing.

The measurements of a flow needs a lot of information e.g. user of the service, path of business services called during this flow, errors returned by service, time parameters, computers executing services etc. In an integrated environment there are usually many services, many flows, many users, etc. It can be easily imagined, that the analysis of stored by a monitor data is time consuming and very complex problem.

4.2 Analysis of the flow results

Analysis of the data stored by the monitoring module makes possible to calculate the percentage of errors returned from the flows or even from a service being the part of the flow. The percentage of errors can be calculated for each user calling the flow. Based on this data the quality of the service (service level) can be calculated. Some errors returned from the flow do not mean that the service is incorrect e.g. incorrect validation in the flow concerned with making up an order. A dictionary, containing all errors which can be ignored in the flow, is necessary for the calculation of the service quality. In figure 2 the percentage of returned errors in some flows is presented. The flows are following:

1. `IVR. x.adapter.ivr:setServiceParams`
2. `IVR. x.adapter.ivr:getServiceParameters`
3. `IVR. x.adapter.ivr:infoService`
4. `SMS. x.adapter.sms:getSmsFromDb`

5. `ECARE_WWW.x.adapter.ecare:serviceModificationOrder`
6. `ECARE_WWW.x.adapter.ecare:getActiveService`
7. `ECARE_WWW.x.adapter.ecare:getAvailServices`
8. `ECARE_WWW.x.adapter.ecare:getContractData.`

On figure 2 some differences in the percentage of errors can be seen. The best results has flow `setServiceParams` (0.4%), the worst – `serviceModifcationOrder` (1.5%). The average availability was 98,5% - 99,5%. The information from this diagram can be used to find low quality flows. These flows are candidates to be improved. Other criteria for choosing a flow for the improvement process are: priority of service, complexity, number of calls of this service, cost. The diagram (fig.2) contains data collected during the whole testing period. Summarized data can be used to find trends. Similar data but gathered in short period of time, may help in finding system bottlenecks.

4.3 Analysis of the flow effectiveness

The stored logs may produce the diagrams showing the effectiveness of flows. The basics criteria is the time to accomplish the flow. In figure 3 minimal, maximal and average time of several flows, listed in section 4.2 are presented.

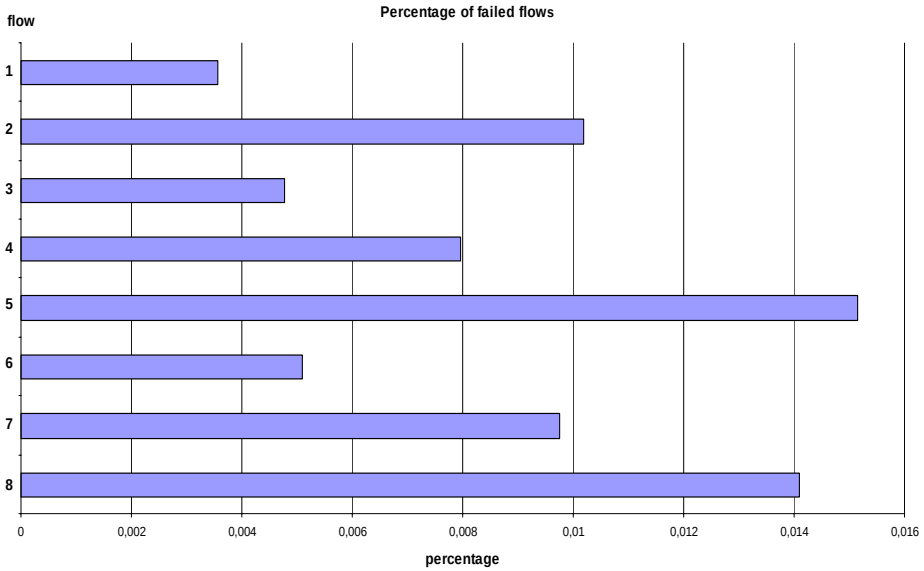

Figure 2. Percentage of errors in flows

If the execution time is crucial for the client, the execution time of the flow can be the basis for the calculation of the service quality. The percentage of the execution times lower than the critical ones, should be compared with values in Service Level Agreement. In fig.3 SMS user calls (4^{th} flow) are significantly slower than others. SMS calls are asynchronically realized because the execution time is not critical for user.

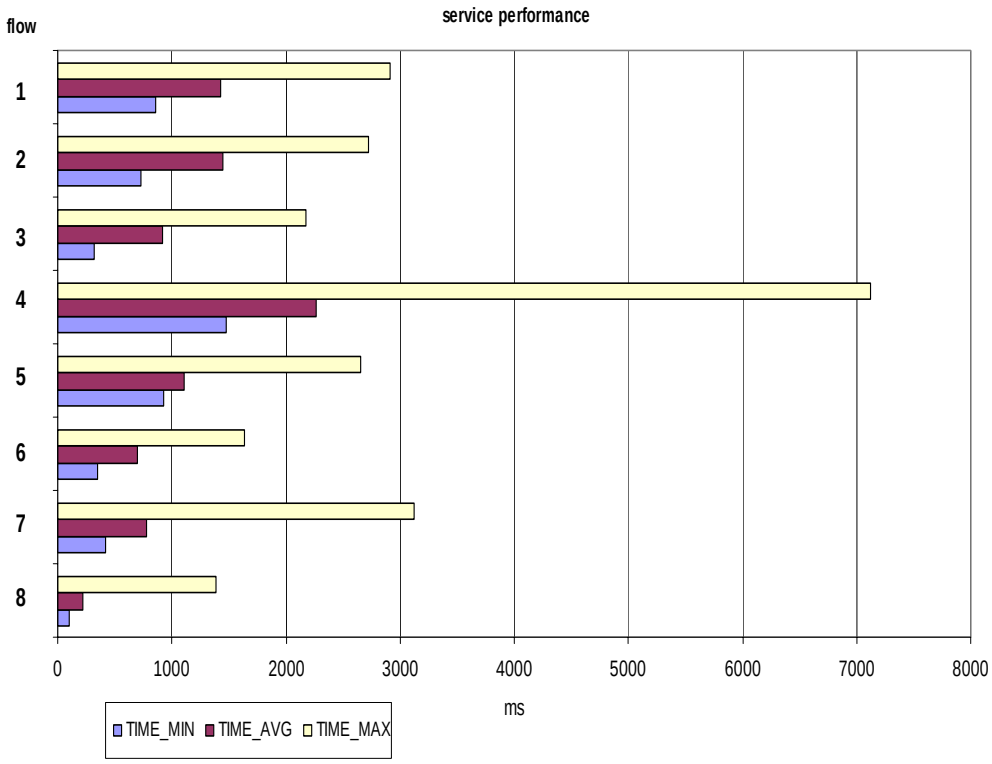

Figure 3. Comparison of effectiveness of flows

4.4 Flows per time unit

The traffic in the integrated system (number of flows realized in a time unit) is very important for system administrator and manager. The flows in figure 4 are following:

1. `x.core.service:getServiceParamsValues`
2. `x.core.service:getAvailServices`
3. `x.core.service:getActiveServices`
4. `x.core.om.service:serviceModificationOrder`
5. `x.core.infoservice:infoservices`
6. `x.core.contract:getContractData`

In figure 4 the percentage of selected flows in total traffic are shown. Such information helps in estimation of the cost of maintaining the integrated system for a client.

In figure 5 the time distribution of these flows is shown. These information are very useful in the estimation of system capacity. System should be able to stand the maximal load (traffic). The data in figure 5 were obtained not during normal service of the systems but during test period, so the distribution of data is not typical.

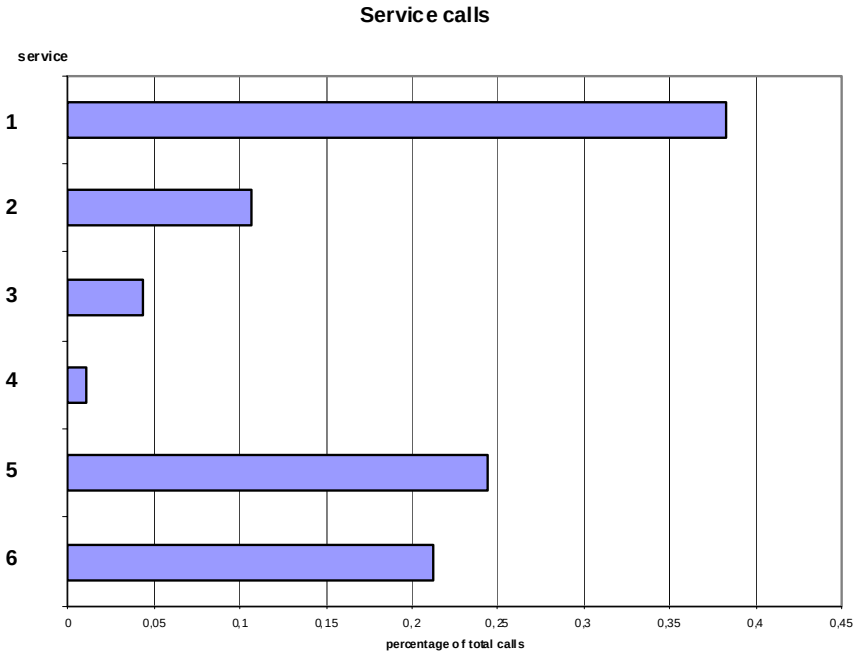

Figure 4. The percentage of six flows in total traffic

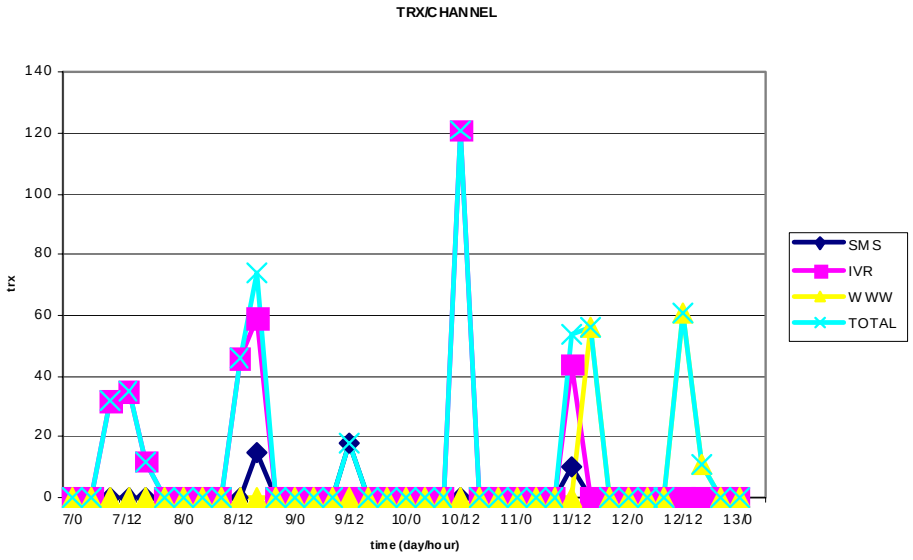

Figure 5. Traffic in the integrated system in some time period

4.5 Analysis of the platform resources

The administrator of an integrated platform should be provided with information concerning the utilization of system resources. The information from section 4.4 is insufficient to calculate the number of servers and connections needed for correct systems operation. Important resources of an integrated platform are the total number of threads executing clients services and connections to external resources. Too high number of concurrently working threads will slow down the server or some services will be rejected. If the number of request to external data bases is higher than the number of connections, services will end with errors. To guarantee the correct platform operation the number of resources needed for the traffic should be estimated. For such estimations the number of flows and the time to execute them on a server is necessary.

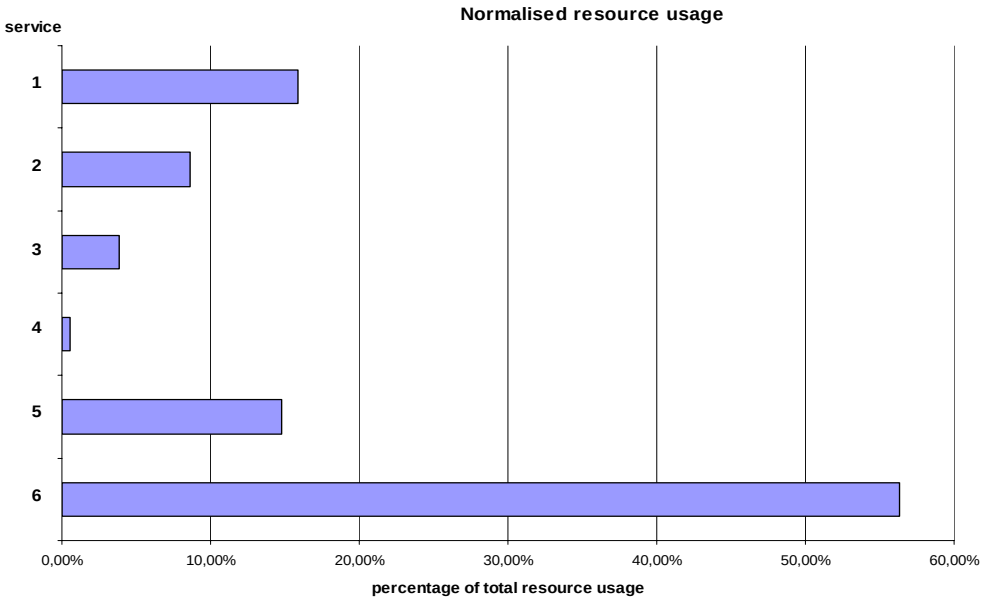

Figure 6. The usage of server resources by selected services

In figure 6 the usage of server resources by selected six services is shown. The names of these flows are given in section 4.4. From this figure the service using maximal server resources can be identified (6th service). This service is a candidate for optimization. If the execution time of `x.core.contract:getContractData` service will be half the current value, 30% of platform resources will be set free. 30% of resources for a big platform is worth some optimizing actions. It may be effective to move some data used by this service to a fast, special data base.

4.6 Server comparison

In large integration platforms it is very difficult to locate a server with some technical problems e.g. network connections, low effectiveness. Such server will increase the execution time of services but the cause of this problem can be hidden for a long time. The data collected by our monitoring module enables the comparison of servers efficiency. For each service the name of server executing this service is also stored (section 3.1). With the information stored in the log data base it is possible to compare the execution time of the same services on different servers. Such comparison can be seen in figure 7. If the servers have the same technical parameters (the case from figure 7) the execution times should be similar. In figure 7 the differences can be seen.

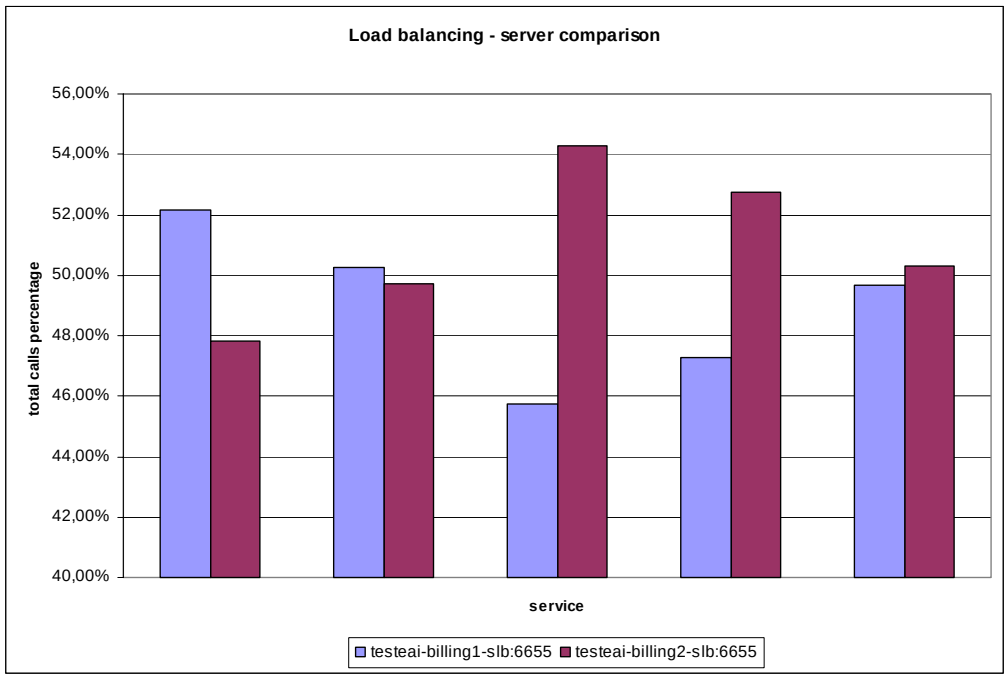

Figure 7. The comparison of servers executing the same service

When we are comparing the efficiency of servers these servers should work with the same load. Even with load balancing the load of servers is not the same and the usage of servers resources is also different. For the two servers compared in figure 7 the load differences were almost 8%. This value can be used to normalise measured values. The normalised performance of servers is presented in figure 8.

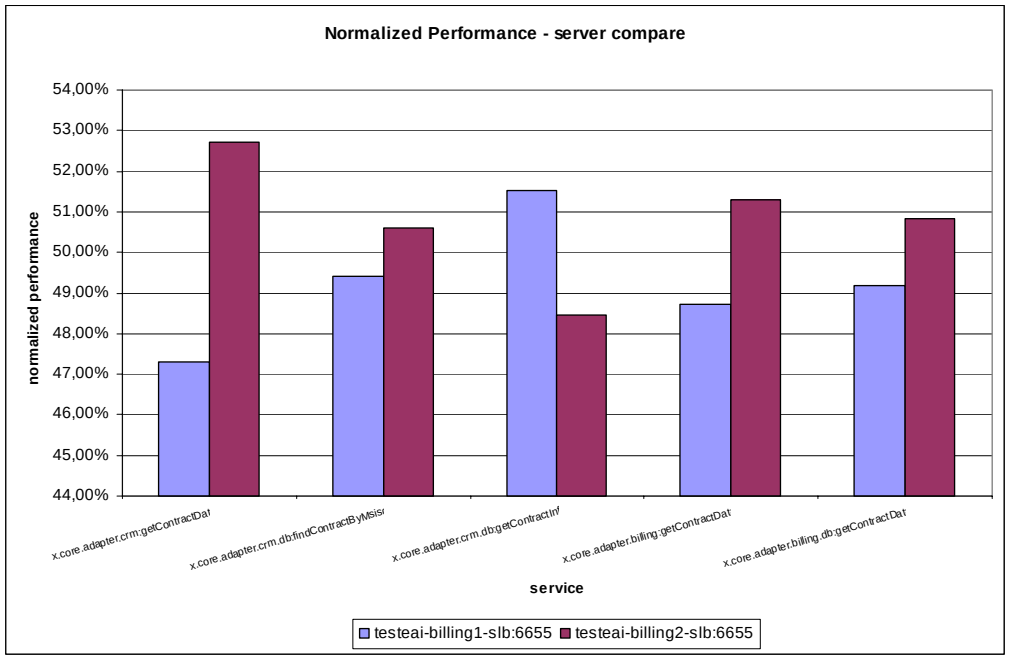

Figure 8. Comparison of servers

5. CONCLUSIONS

The quality of services on an integrated platform is influenced by many factors: e.g. the physical and logical architectures, integrated systems, implementation of flows, technology, etc. Finding bottlenecks or services of low quality is a very difficult task. The monitoring module implemented at the Institute of Computer Science Warsaw University of Technology described in section 3 and in [9, 2] can be very helpful. This module is dedicated to webMethods [10] environment. The idea of monitoring services in SOA architectures, implemented in this tool, is general so the design of this tool can be easily implemented for others SOA integrating products. The data collected by monitoring module can be used by administrators and managers to improve the quality of service on the integrated platform. Examples of such analysis are given in sections 4.2 – 4.6. The monitoring module may be also used to detect bottlenecks of the integrated systems. Such measurements should be performed under normal, typical load of the system (not during functional testing as described in this paper). The module may be used to calculate the quality of services, the utilization of system resources, to observe the traffic on the integrated platform.

The module is able to monitor services on the ESB (Enterprise Service Bus) level even, if the services are executed on different servers. The whole context of the flow is provided to the user. Such information may be very useful in the maintenance of integrated systems. After changes in some services, the monitoring module will collect data enabling the administrator to tune the system, to find services which should be optimized. The majority of existing moni-

toring modules is able to monitor business processes e.g. Optimize for Process in webMethods. Other programs monitoring ESB, are dedicated to only one server or do not provide sufficient information if the execution is distributed on several servers. The overhead of the monitoring module was measured [9]. For a flow consisting of 100 services the overhead was 12 to 31 msec. This is an acceptable value and lower than the overhead caused by other integration software.

Further work will concentrate on integration the monitoring tool with other tools enabling the generation of reports e.g. BMC Patrol [3], or HP OpenView [8].

REFERENCES

1. Baresi L. et al, 2004. Smart Monitors for Composed Services. *ACM ICSOC'04*. New York, USA, pp. 193-202.
2. Bluemke I. and Warda M., 2007. Monitoring module (in polish). *Proceedings of Second Polish Conference on Data Processing Technology*. Poznań, Poland, pp. 534-545.
3. BMC Patrol, 2007. `http://www.bmc.com/` .
4. He H., 2003. What is service-oriented architecture? in http://webservices.xml.com/
5. Kumar P., 2005. Web Services and IT management, *ACM QUEUE*, July/August, pp. 44-49.
6. Mahbud K., and Spanoudakis G., 2004. A Framework for Requirements Monitoring of Service Based Systems. *ICSOC'04*, November 15–19, New York, USA, pp. 84-93.
7. Marchetti C., et al, 2004. Quality Model for Multichannel Adaptive Information Systems. *WWW2004*, New York, USA, ACM 1581139128/04/0005, pp. 48-54.
8. OpenView 2007. `http://h20229.www2.hp.com/` .
9. Warda M., 2006. Architectures for Enterprise Application Integration. MSC. Diploma (in polish), Institute of Computer Science Warsaw University of Technology.
10. webMethods, 2007. `http://www.webmethods.com` .