EFFICIENCY OF ENTERPRISE SERVICE BUSES – COMPARATIVE ANALYSIS

Artur Bobrowski, Tomasz Walkowiak

Institute of Computer Engineering, Control and Robotics
Wroclaw University of Technology
ul. Janiszewskiego 11/17, 50-372 Wroclaw, Poland
E-mails: artur.bobrowski89@gmail.com, Tomasz.Walkowiak@pwr.wroc.pl

The paper describes the analysis and discussion about the efficiency of selected enterprise service buses. A few test scenarios when systems communicate with each other in different manners are developed. IBM WebSphere ESB, Mule ESB, WSO2 ESB has been chosen to verify their efficiency. Each of them represents different attitude to licensing process so they are good testing samples. Implemented scenarios are tested from two points of view – throughput and latency. These are the most common measures used to describe the system availability and its efficiency. They allow us to characterise system’s behaviour in the critical situation, when peak of incoming messages occurs, and define architecture’s impact on message processing times when system performs regular activities. Moreover, this paper describes methods of executing performance tests, their results and comparison with other researches. The analysed measures often are essential for the owner and administrator of the complex distributed systems.

Keywords: efficiency analysis, performance, enterprise service bus

1. Introduction

Nowadays there is observable trend to exchange data between systems. This demand is caused by growing necessity of using many different systems which were created some time ago. Thanks to that it is possible to take advantage of data collected by these systems by now. Moreover, using finished third-party systems lets companies to reduce work and cost needed to develop their own solutions that provide the same functionalities. These factors caused appearance of so called service integration. Currently, service integration gains more and more significance. Its general purpose is letting two different systems to communicate with each other. This communication consists in defining special contract that makes possible exchanging data between sides. One side that prepared services and made them accessible by others is called service provider and others that are using those services are called consumers. Such approach lets the enterprises to concentrate at their own specific requirements which are needed for their commercial and business activities. In addition using services published by other providers brings about higher reliability and confidence in supporting standards too.

Service integrations is not an easy task though. Before its initiation it is required to make a detailed revision of service interfaces used by all systems that are going to take part in integrated communication. This step allows adapting these systems by preparing message transformations that adjust data model used in communication to that one which is used internally by system. The more systems are trying to communicate with each other the more complicated is the task to integrate them. That is the reason of inventing enterprise services buses. They have built-in mechanisms that simplify integration and service publishing processes. By making the market analysis, it is easy to notice that enterprise service buses are used by more and more companies. Their potential was appreciated and they make modern systems more flexible and scalable. Conducted researches [14] confirm that enterprise services buses are currently used or will be introduced by most of asked enterprise system architects and application development managers. Message routing and transforming are two mostly used features of enterprise services buses but these are not the only functions provided by them. Choice of the solution used in company is difficult and it depends to a high degree on specific characteristic and needs of the project [12]. Additional influence on decision has also a main purpose of enterprise service bus using. Class of problem, realised patterns of communication between systems and implemented business logic implies expected technical abilities of chosen product [15]. Because of the rising popularity of the enterprise we decided to examine chosen enterprise service buses with respect to one of the most important criteria – their efficiency. Measures which we wanted to verify were throughput and latency. Their characteristics are described in detail in the Section 2.1.

The organization of this paper is as follow. We start with overall review of carried out studies and describes theoretical aspects of enterprise service buses efficiency (Section 2). The chosen test scenarios were base for an implementation in selected environments (Section 3) and implied the required elements that were shared between all checked solutions (Section 4). Also, we describe our proposal of conducting experiments and measuring the efficiency (Section 5). Next (Section 6), we present the results of our experiments.
2. Related Work

This chapter describes issues related to efficiency of enterprise service buses. It explains how the experiments should be conducted and what values can be measured to characterise particular solutions. Moreover, there is a subsection that refers to related works.

2.1. Efficiency interpretation

According to [6] there are many different ways to test enterprise service bus efficiency. There is one general approach to this problem though. During efficiency tests, there are three distinguished parts of the whole system. The main part is obviously enterprise service bus which is a central point of the system. Other parts are also very important, because their duties are generating incoming messages for enterprise service bus and creating responses which are sent back to client by the service bus. Their roles are the same as service consumer and provider in real environments. They are called respectively load driver and dummy service. From this point of view there are three possible ways of conducting experiments. Each test can operate at different enterprise service bus as the main part of the system. This approach is used when one wants to compare capabilities of various solutions created by different producers. On the other hand when it is important to examine specified solution thoroughly experiments are taken with different configurations of enterprise service bus and differ in realised communication pattern. Another variant is based on comparison between system’s performance with direct communication and during using enterprise service bus as a mediator.

Generally there are two expected results of efficiency testing. From the users’ point of view very important measures of efficiency are throughput (requests/second) [7] and latency (how long each request is processed by enterprise service bus) [18]. These metrics cause two different ways of message generating. To measure latency brought up by enterprise service bus the load driver is implemented in a way that simulates real load that can be observed in the running system, i.e. one hundred users that sends one request each of fifteen seconds. On the other hand, to measure maximum throughput that can be withstood by enterprise service bus it is needed to simulate very high load. This scenario is called saturation and it is implemented as many users (which means much more than during the latency measuring) who send another request immediately after receiving response to the last one.

Regardless of chosen type of load generating process there is expected behaviour pattern of enterprise service bus efficiency. This pattern is shown on Figure 1. This figure represents throughput and latency dependence on rising load. At both charts there is noticeable point when the line changes its trend. In case of throughput this moment occurs when the enterprise service bus is not able to process more incoming messages and starts to buffer them. After this point the throughput does not change and stabilizes at maximum level according to product capabilities. Unfortunately, as explained in [6], it is ideal scenario, which has never been found in reality. The real throughput after exceeding the critical point drops sharply. Latency shows different behaviour. Initially, it is almost constant without any affection of rising workload. At some point it starts to increase linearly, which is natural and is caused by more incoming messages, which need more time to proceed. However, while exceeding the critical point, the observed latency shows tendency to rise very quickly (i.e. in exponential manner).

2.2. ESB Performance reports

Due to growing popularity and usage of the enterprise service buses their efficiency benchmarks became very common discussion topic. Their executors are mainly enterprise service buses’ producers who are trying to prove their solution superiority over other solutions. There are also independent people who
conduct their own tests because of their curiosity. The most well-known benchmarks of enterprise service buses are reports published by AdroitLogic Company called ESB Performance. These reports are published once a year and are called rounds with number incremented by each edition. This time span allows showing results of changes implemented in new versions of analysed solutions. The newest report is called Round 6 as it is sixth edition of this benchmark. Both results of these benchmarks and tests’ implementations are open and available for everyone because creators expect comments to test scenarios and corrections of implementation and configuration. Because of that each round slightly differs from each other. In ESB Performance Round 6 there were carried out 6 different test scenarios: direct proxy, content based routing proxy, SOAP header content based routing proxy, transport header content based routing proxy, XSLT proxy and WS-Security proxy. These scenarios are the most common in the real environments created by companies to realise their service integration logic.

WSO2 company that is a creator of the WSO2 ESB solution wanted to improve image of their product that achieved quite poor result in sixth edition of ESB Performance did not want to wait for the next official benchmark and executed their own experiments based on resources published by AdroitLogic company. The reason of this haste was release of a new version of their product. They called these experiments Round 6.5 as it was not official benchmark. WSO2 with this benchmark wanted to present the high improvement of their enterprise service bus. In comparison with the last official benchmark solution created by WSO2 company took over lead in almost all defined test scenarios.

Described benchmarks in spite of variety of conducted experiments do not take up a subject of asynchronous communication. Moreover, they are based only on non-commercial solutions that are available in the market. These facts were reasons to conduct our research which included the aspect of asynchronous messaging and comparing well-known free solutions with commercial one.

3. Test Environment Preparation

This chapter contains specification of the defined test scenarios, which were used to measure efficiency and short description of selected enterprise service bus solutions.

3.1. Test scenarios

Within the framework of this research it was necessary to define scenarios, which should be implemented to examine selected enterprise service buses and compare their capabilities. Authors decided to include as much as possible mediation patterns in these scenarios. Thanks to that it was possible to verify difficulty of creating applications and capabilities of enterprise service buses. All test scenarios that was proposed were defined in the simplest manner and it is very unlikely to spot such applications in the real systems’ environments. Nevertheless, these scenarios were based on real applications so it is probable that their variations may be used somewhere where particular functionality is needed. Obviously, proposed scenarios were common for all tested enterprise service buses.

After analysing the most common usage variants and taking into consideration matters mentioned earlier in Section 2 authors decided to propose the following scenarios:

A. request-response communication pattern (synchronous invocation)

It is the most often used pattern of communication between systems in service oriented architecture environments. This scenario’s purpose is allowing us to exchange data between two systems that use different data models and interfaces. The main purpose of this scenario is to verify abilities of enterprise service buses to transform incoming requests and received responses. That task is essential to make communication between those systems possible. Besides of that, this scenario should also include message monitoring. This task has been realised by logging message content alongside with other message processing related information to the log files. Logging is performed in four points – before and after each (request and response) transformation. Such approach makes possible carrying out audits that lets to determine whether the enterprise service bus system and applications are working properly and to detect abnormalities earlier.

B. request only communication pattern (asynchronous invocation)

This is rarely used communication pattern commonly applied in services that need much time and/or resources to complete their processing. Such services’ responses are usually available as the result of other operations’ calls. Example of such situation may be the periodically generated reports (such as monthly balance sheet). In this case it is properly to create two service operations: i.e. asynchronous prepareReport() and request-response retrievePreparedReport() with additional parameters to correlate these calls. This scenario should be implemented in a way that incoming request is wrapped and saved in the message queue. Because of usage of message queues it will be
necessary to carry out the protocol switch. Before saving message should be enriched with additional information that sets processing priority of this particular request.

C. service gateway (message routing)

This scenario is the most complex among the chosen scenarios. It is used in very advanced environments where it is necessary to integrate many services with each other [11]. Usually, systems have hierarchical structure so their connection is even more complex. Service gateway is used to receive all messages passing through the company’s network and dispatch them to appropriate systems. Because of that, service gateway is one of the best places to apply the security policy. Service gateway often is connected with the service registry but the dispatching rules may be stored in other external resources such as databases or files. Because this scenario is very specific and rarely used in practise, authors decided to implement simplified version of this scenario. In this case the main task of the applications is the content based routing (CBR) [13]. This variant is usually applied when the executed business logic is dependent on certain conditions. Dispatching the incoming request to the proper service provider is realised on the basis of message content.

3.2. Selected enterprise service bus solutions

To satisfy the needs of this research, it was necessary to choose a few enterprise service bus solutions which would be compared with each other. To assure the objectivity of this research it was essential to test a wide spectrum of available solutions. Taking all these factors into consideration, authors decide to select three solutions that differ with their attitude to the users.

Because in the ESB Performance reports there is not present any commercial product, the first of the selected enterprise service buses is the IBM WebSphere ESB Registry Edition1. According to IBM Company licensing system license is issued on the basis of physical resources instead of working solutions. These licences limit usage of provided applications to the number of cores per license. Support from the producer to the owner company is additionally paid and it requires cooperation between licence’s owner and IBM Company. These aspects are reasons that this solution is rarely used in small, simple projects and usually is preferred by composed organizations with high budget. In return for high prices of this solution users acquire assurance of well-known company reputation and high reliability.

The second choice of the enterprise service bus system is Mule ESB2 manufactured by MuleSoft Company. This solution is available in two different versions: free Community Edition and paid – Enterprise Edition. Enterprise Edition allows using more built-in features of this enterprise service bus. Moreover, after purchasing the paid variant of this solutions users’ gains additional support from the producer and faster response in case of any problems caused by errors in this enterprise service bus.

Finally, the last chosen solution is product of the WSO2 Company – WSO23. This product is totally free of charge. Only additional support from the WSO2 Company to the users requires payment. This solution is based on a few projects led by Apache Software Foundation. Main technology used to create and run applications in this environment is Apache Synapse. Its guidelines are kept and WSO2 ESB only provides runtime environment, consistent with other products from this company. Such approach guarantees integrated environment (the so-called ecosystem) when each product is managed in the same way. Thanks to that it is much easier to plan, prepare, implement and control complex systems, which are based on similar solutions made with the same view of basic concepts.

4. Shared Elements and Implementation

In order to obtain as reliable and authentic results of comparison as possible alongside with testing environment authors prepared also the set of data types and interfaces. This data model has been used in each scenario implementation and is the same for each tested solution. Prepared data model was based on hypothetical company that published three web service operations. Data types represented the following objects: customer, product and order. There was also created additional type CustomerLegacy that represented customer object used by legacy system, which needed to be integrated with other systems. The only difference between types representing customer were different name and type of identity field. Current model uses integer value with field name ‘id’ while old version expects field with name ‘customerId’ and string type. All defined data types are simplified to the limits and contain only a few fields that allow telling the instances apart.

2 http://www.mulesoft.org/
3 http://wso2.com/products/enterprise-service-bus/
During the experiments’ preparation phase there were being created two web service interfaces. The first of them is called EsbInterface as it is known to applications in enterprise service bus runtime and published by them to the other systems. The second one has names ExternalInterface as it represents web service operations that uses different data model than enterprise service bus and is published by other systems. Both interfaces were created in accordance with WSDL 1.1 standard and document-literal encoding [2]. There were the following defined operations: getCustomer, addCustomer and prepareOrder. Their specific usage may be described as follows: getCustomer allows the external service consumers to use data stored in legacy system thanks to integration with enterprise service bus interface. addCustomer is asynchronous operation, which is called at published interface and dispatched to specific system’s message queue with additional information about priority of request processing. Last but not least operation prepareOrder after receiving by enterprise service bus is dispatched to one of the available service providers on the basis of message content – in this scenario decisive factor is whether customer has premium status.

One and only external resource shared between all solutions was JMS service provider. Its usage was necessary to accomplish accordance with scenario B specification. Authors decided to use Apache ActiveMQ\(^4\) product made by Apache Software Foundation. It is one of the most often used solutions and its popularity is its greatest advantage. Because of large community connected with this solution it is very easy to implement applications and resolve coming out problems that are related to this JMS service provider. Moreover, this solution is mature and provides wide-range abilities. These factors are the reasons why this solution is very often used not only in small and academic projects but also in complex solutions when high reliability and efficiency are crucial factors.

5. Experiments’ Methodology

On the basis of the issues presented in Section 2.1. efficiency measuring is not an easy task. Additional difficulties are mentioned in [7] that take up a subject of enterprise systems’ efficiency testing. Even though it is not the most-recent published paper, the thesis, which has been introduced in it, still is up-to-date and valid. It pictures the problem of efficiency measuring in very general and understandable way. According to it the most important part of such research is deciding what result one wants to obtain and/or what metric he or she wants to measure. These decisions should determine the test system architecture. It is also important to keep in mind that simple comparing results with other available researches in a quantitative manner is not right because the outcomes vary in latencies and throughputs because of different solution configurations, tested scenarios, application implementation and used hardware. Such problems are also described in articles mentioned earlier.

According to [6], it is essential to separate responsibility for required task between different systems. Enterprise service bus class solutions were the main part of test environment architecture. It was also required to prepare dummy service, which is able to generate responses that enterprise service bus should pass to the service consumers. Authors selected the SoapUI solution [10]. It is very creditable solution used to test web services. Despite the fact that it is available free of charge, its range of abilities is very extensive. Apart from web service testing this application allows us to create mock services based on WSDL interfaces. These mocks are available from external applications and can generate responses adequate to received requests. Mocks created to fulfill requirements of test scenarios are designed in such manner that allows usage of Groovy scripts. These scripts are added to increase variety of generated responses. Thanks to that the possibility of caching responses by enterprise service buses are reduced to minimum. The responses are generated with dynamic content based on fields’ values in incoming request that is being processed at that moment.

SoapUI application was also used as unit test tool to conduct basic functional tests. Their purpose was to determine if logic implemented in enterprise service bus solutions as well as mock’s Groovy scripts worked properly and in accordance with the specification. However, the main testing tool was Apache jMeter\(^5\). This application is designed to provide an advanced testing environment. Furthermore, there are many plugins that enables additional functionalities. During this research we use jMeter Plugins set, which provides many useful configuration elements. One of them is load generator that can be configured to generate load in time function. The general course of active threads that simulate users is presented on Figure 2. In the beginning the number of active threads rises steadily and at the determined point stabilizes. Then requests are sent to enterprise service bus in a continuous manner. Such conditions are kept by some time and after that period threads are gradually stopped. Thanks to that it is possible to simulate more authentic load because in this way requests are not sent simultaneously but with different time intervals between each other. Moreover, the enterprise service bus has time to allocate required amount of physical resources to process all incoming requests.

---

\(^4\) http://activemq.apache.org/
\(^5\) http://jmeter.apache.org/
Dashed lines on Figure 2 delimit time period when requests have been taken into consideration. They were extracted from all logged request by simple Java application that parsed JTL files (jMeter log files) in order to reduce range of request, which was taken into consideration during evaluating efficiency measures.

Apache jMeter test plans are saved as XML files so it is possible to prepare another Java application that modifies parameters of test plan. Thanks to that all tests could be run automatically by PowerShell script in the following sequence: modify test plan parameters (i.e. destination endpoint, active users, etc.), run jMeter test in non-GUI mode, extract requests from log files to compute expected measures.

![Figure 2. Load generated by Apache jMeter during single efficiency test in time function](image)

Each of tested enterprise service buses was tested with running profile set as standard settings without any modifications except the JMS service provider connection configuration. The main purpose of this approach is checking abilities of enterprise service buses to work in the highly loaded environment shortly after installation. Thanks to that, it is possible to verify capabilities of the tested solutions in the configuration prepared and proposed by producer, which should be the most universal and provide high adaptability and flexibility. It is important to notice that all the results obtained during research and presented in this paper are not definitive and it is possible to improve them by strenuous and long-lasting profiling process and adjusting configuration parameters to the specific project’s requirements.

All results presented in this paper are the effect of filtering and averaging the results of prepared test plans for each of test scenarios. Tests were conducted on the following additional assumptions:

- throughput will be measured with the load generated by number of active clients equals from 50 to 400 with the 50 clients step,
- latency will be measured with the load generated by 200 active clients who sent defined number of requests per second sent by all clients – tested values were in the range from 5 to 50 requests per second with the step of 5 requests per second.

6. Results and Discussion

This chapter contains results of executed tests aggregated in the form of charts depicted on Figure 3 and Figure 4. Results are presented as the line charts with division made in a way that gives a set of measure and scenario combination pairs. Each graph represents all obtained results in the specific scenario by all tested enterprise service buses and for all input parameters.

On the basis of the presented results, it is possible to draw the following conclusions:

- The most noticeable fact is that IBM WebSphere ESB got the worst results in all tests among tested solutions. Reason of that may be the fact that product of IBM company is the only one that is not free of charge and is fully commercial. By being commercial, we mean that it is mainly directed to large companies with very high budgets which can effort high costs of licences. That fact determines that it needs much more physical resources to operate properly. Unfortunately, during this research IBM WebSphere ESB operated at machine that had only 4 GB of RAM which was shared between ESB runtime environment and DB2 database used by application server. Therefore, this is very likely main reason of lower values of throughput and higher latency.
- The remaining two products: Mule ESB and WSO2 ESB showed similar efficiency. Their results were highly dependent on test scenario. The highest difference between characteristics of these
solutions can be observed on Figure 4c, which represents results of latency measuring for scenario C. In this case latency generated by WSO2 ESB suddenly started to rapidly increase when generated load reached level of 40 requests per second.

Figure 3. Throughput in function of active clients for a) scenario A, b) scenario B, c) scenario C
Figure 4. Latency in function of generated requests per second for a) scenario A, b) scenario B, c) scenario C
The most noticeable differences between tested solutions are observed in scenario C. Throughput characteristic differs significantly for each of products. Such differences are caused by usage of various kinds of libraries used by each of the solutions to analyse the message content. This is the feature of used scenario because content based routing is depended on XML processing to a high degree. Enterprise service bus efficiency results are the outcome of efficiency of each of internal elements: XML document parser, object representing method and other related to processing messages with knowledge of their contents and expected structures.

In the scenario B tested enterprise service buses achieved relatively lower efficiency. Higher latencies and lower throughput are caused by necessity of transport protocol change. In case of this scenario, as opposed to scenario A, except message content transformation it was necessary to modify whole processing context. This operation included adding new JMS transport headers which was used to save additional information alongside with message content in the message queue. The final results are worse than in other scenarios despite the fact that communication with service consumer needed less amount of resources because as a response there was sent only HTTP protocol code 202, which meant that request was accepted without any response body. It was important to remember though that tests were executed in the local network environment, so delays introduced by physical network topology were close to zero, so in the real environment when consumer was not connected to the same network as enterprise service bus the outcome efficiency might vary.

Considering test environment settings the best efficiency was reached around 150 to 200 active clients. Mule ESB was the product that kept relatively stabilised throughput close to optimal values. After reaching critical point the throughput drop is also the smallest in relation to absolute value among tested products. Exception to this rule is scenario C throughput test. In this scenario WSO2 ESB had not only higher throughput but also kept it at stable level longer. The critical point where efficiency collapsed occurred at the level of 250 active users and it is the best result among tested enterprise service bus solutions.

Efficiency characteristics obtained from the conducted experiments are coincident with the empirical behaviour that has been described with details by the WSO2 company in [16]. With the increasing number of active clients the throughput is rising accordingly till the critical point is reached. After that point efficiency drops and more rejected incoming connections to enterprise service bus are observed.

Latency characteristics are consistent with other results. When generated load is not big then latency added by enterprise service buses is at stable level. After that we can observe point when latency is starting to rise but increase is gradual and linear. This situation remains till the critical point. At that moment enterprise service bus is not able to process all incoming requests in appropriate amount of time. After reaching this point latency is increasing faster in an exponentially manner. The reason of that behaviour is greater time of processing each request and higher time intervals used to wait for available resources between processing phases in message pipelining process.

Obtained results are dramatically different with respect to absolute values of presented measures than results showed in ESB Performance reports [1][17]. That is caused by differences in hardware configuration used to execute tests. However, in relation to qualitative characteristics one can notice similar regularities in case of Mule ESB and WSO2 ESB results both described in ESB Performance reports and observed during this research in scenarios A and C. In those scenarios better results have respectively Mule ESB and WSO2 ESB. This is consistent with their results presented in those reports for similar test scenarios.

ESB Performance reports do not contain any information about IBM WebSphere ESB so results obtained during this research cannot be confronted with any other available references. Other element which is not mentioned in ESB Performance reports is verifying enterprise service buses capabilities in the field of asynchronous communication. On the basis of executed experiments it can be observed that among tested solutions the best results obtained Mule ESB. Its supremacy is especially noticeable in case of number of requests processed per second. Moreover, higher throughput is achieved alongside with latencies at similar level as generated by WSO2 ESB. So good results of these enterprise service buses may be caused by usage of Apache ActiveMQ JMS service provider. This provider is one of the recommended solutions by producers to connect with their products. Connection configuration process for this provider is described in details in the documentation published by producers. Moreover, Mule ESB provides dedicated connectors with predefined connection configuration, which requires only a few additional parameters. As opposed to these solutions, IBM WebSphere ESB does not provide any dedicated connectors and users have to configure connection with this provider as a general JMS provider. That configuration is very complex and gruelling process, which requires creation of many additional objects that are used during communication between JMS service provider and enterprise service bus. Such communication overhead is probably the reason of lower efficiency of this solution.
7. Conclusions

We have proposed a model and methodology of enterprise service buses efficiency and presented results obtained during conducting our experiments. Efficiency is only one of the factors that determine whether the solution is suitable and meets requirements. This is caused by differences between applications, systems and their architectures. It depends on specific project whether the most important factor is efficiency or rather users prefer higher reliability, flexibility or easiness of development and management process. As these factors may be also very important it is necessary to make detailed market analysis containing all elements that are crucial in case of actual project.

Testing methodology that has been proposed by the authors is based on other studies performed in the field of service oriented architecture that connects with the enterprise service buses efficiency. Our test scenarios and measuring process were preserved in a way that was consistent with well-known rules and principles of efficiency testing. Moreover, thanks to usage of very popular and advanced third-party testing tools it was possible to prepare automatic testing environment, which was very flexible and could be adjusted to other products and/or scenarios in an easy manner.

Results obtained during this research are not definitive as it has been mentioned in this paper already. If one spends enough time to adjust configuration and match its parameters with specific needs then there is a high probability that used solution will achieve better efficiency statistics and satisfy project’s requirements to the highest degree.

The presented work has been supported by the Polish National Science Centre under grant no. N N516 475940.

References