

An Integrated Broker Platform for Open eHealth Domain

Foteini Gr. Andriopoulou, Lamprini T. Kolovou, and Dimitrios K. Lymberopoulos

Wire Communications Laboratory, Electrical and Computer Engineering Department,
University of Patras,
University Campus, 265 04 Rio Patras, Greece
fandriop@upcatras.gr, lamprinik14@gmail.com,
dlympero@upatras.gr

Abstract. The adoption of personalized context aware services in healthcare domain has imposed new demands on IT providers and motivates integration and interoperability between heterogeneous healthcare systems. In this paper, we propose an Integrated Broker Platform – IBP that incorporates the benefits of the advanced technologies of Enterprise Service Bus (ESB) and Service Broker (SB) allowing the efficient provision of secure, interoperable, reliable and cost-efficient message and service delivery by dynamical and intelligent selection of services. IBP provides mediation functionalities that TSB supports but is enhanced with business logic from the SB. An architecture pattern for both TSB and SB is proposed, analyzed and prototyped.

Keywords: integrated broker platform (IBP), ESB, TSB, Service Broker-SB, healthcare.

1 Introduction

Nowadays, healthcare systems rapidly moved from treating isolated episodes towards a continuous treatment process involving multiple healthcare professionals and various healthcare infrastructures (e.g. hospitals, clinics, institutes). This rapid change in the healthcare domain imposes new demands on IT providers and motivates integration and interoperability among heterogeneous software components within the health information systems [1]. Integration and interoperability of different, heterogeneous software components, however, is a difficult task, as applications usually are vendor proprietary and not designed to cooperate with other vendor applications. Today powerful integration tools (e.g. application servers, object brokers, different kinds of message-oriented middleware, integrated platforms, Enterprise Service Bus (ESBs), Service Delivery Platforms (SDPs), etc) are available to overcome the heterogeneity of system components [2].

ESB is an evolution in the integrated middleware software architecture that has gained the attention of architects and developers, as it provides interoperability, integration, mediation, security and reliability. The main role of the ESB is to serve as a communication bus accepting a variety of input message formats and transforming them to different output formats and thus providing a transparent communication

interface. Nevertheless, ESB implementation itself is not standardized but offers a messaging infrastructure based on standardized protocols. As a result, there are major differences in the feature sets of available ESBs (e.g. Oracle, IBM, Microsoft, Nokia, Siemens, etc) as vendors try to differentiate from each other.

In the eHealth domain, some enterprises have proposed and implement healthcare integration solutions such as IBM, Microsoft that are fully vendor proprietary and based on web service technologies. Moreover, some research efforts are trying on to create open ESBs for healthcare purposes but all of them focus on web services technologies. In [3] L.Gonzalez et al have proposed an e-health integration platform that is based on semantic and web service technologies for social security services. In addition, S. Van Hoecke et al in [4] have proposed a user- friendly and secure broker platform for e-homecare services that was designed using web service technology. It provided a well established mechanism for authenticating user once and being always connected, also for the implementation used an open ESB for the integration between requestors (users) and providers. Nevertheless, in the Hoecke proposal the syntactic, semantic and ontology integration is still a complex and complicated issue. Moreover, the Open eHealth Foundation [5] has already proposed and implemented an Open eHealth Integration Platform (IPF) that is based on open standards and Apache Camel [6]. Since IPF is licensed by Apache Software Foundation [6] the whole implementation is based on Apache software products and tends to be vendor proprietary even if Apache supports open source projects.

In this paper, we propose an Integrated Broker Platform (IBP) for open eHealth domain that supports the mobility of citizens and caregivers by providing the integration of various services through a unique framework. We propose the IBP architecture pattern and the functionalities that an open platform should have so as to let any developer and enterprise to create their own or use proprietary and legacy technologies. IBP provides syntactic and semantic interoperability by performing related transformations and is based on open standards in order to achieve the interoperable cooperation of different communication protocols and interfaces. IBP is a combination of cooperating but autonomous brokers. Each broker has its own functionalities and mechanisms so as to be completely autonomous and independent of the others in order to support a scalable and horizontal architecture. The autonomy and independency of each single broker provides the openness of IBP. Moreover, IBP is characterized as intelligent since it routes intelligently the messages to the appropriate brokers and provides an intelligent way to find/ bind and invoke services. In this paper, we demonstrate a fundamental IBP structure with a message and a service broker. The proposed architecture tends to bring interoperability both in personalized and mobile services [1].

2 Functionality of the Proposed

2.1 An overview of the IBP

The IBP for eHealth domain enables integration of services and data, and ensures interoperability between different proprietary devices, software systems, operating

systems and implementation languages. IBP aims to simplify user interactions, provide security and guarantee quality requirements (e.g. QoS, QoE, etc).

The proposed IBP, as presented in Figure 1, is composed of: (a) a Telemedicine Service Bus (TSB) that is used as a backbone broker providing efficient message transformation and intelligent message routing [2] and (b) a Service Broker (SB) acting as the integration engine for finding/ binding and invoking of the requested service.

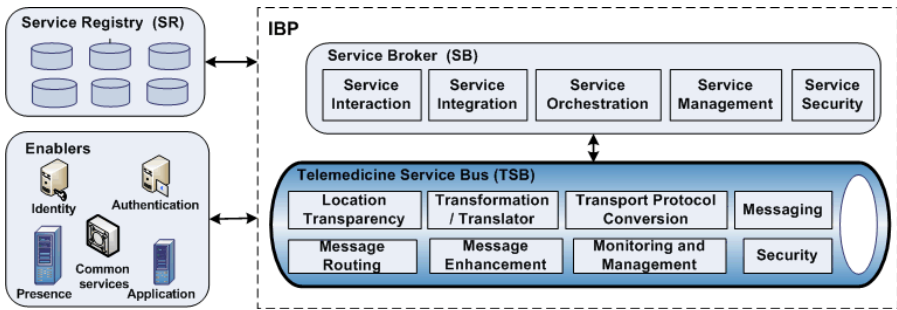


Fig. 1. An overview of the IBP

IBP is activated by event messages from various Enablers [2, 7] that acquire context aware data and provide operational events of any personalized healthcare case (e.g. elderly and disable people). Internal mechanisms of Enablers (e.g. control, administration mechanisms, etc) create standardized messages for the communication of these data and events with IBP through standardized APIs [Parlay/ETSI [8]]. These messages contain a Header and a Content (Body) part; however this message structure is out of the scope of this paper.

TSB is the receiver of the incoming message forwarded by the Enablers. Actually, TSB is a message channel, constructed as a common open source ESB based on Service Oriented Architecture (SOA) and enhanced with protocols for healthcare domain (e.g. DICOM, HL7, CDA, etc). The basic functionality of TSB is to mediate the messages from the Enablers to the SB [2, and 7]. The SB receives the transformed messages from TSB and provides mechanisms for processing the content of the messages in order to find/ bind and invoke the requested service from a Service Registry (SR). The SB is enhanced with open standard interfaces for sending and receiving messages from TSB to 3rd party providers and applications. These interfaces are combined with a set of strong rules in order to guarantee message delivery and processing [2, 7]. Finally, one or more SRs contain the lists of the services and URLs provided by 3rd party providers so as their services and resources to be accessible and discoverable from the SB [2, 7].

2.2 IBP Fundamentals

The proposed IBP is a brokering platform that provides end-to-end communication service through independent and operating autonomous brokers (TSB and SB).

In order to provide the full functionality character of the brokering platform, both the TSB and SB support the basic functionalities of message transferring, storing and verification.

- *Message Transferring*: The service components (TSB, SB) are responsible for transferring, handling, addressing, identifying and converting messages from the Enablers or from the Service Registry. For the efficient and secure transferring of these messages it is essential to manage and negotiate user's capabilities and relevant security issues.
- *Message Storing* is provided by service components such as data stores that handle the transmission and receipt of messages from or to the message storing entities.
- *Resource Verification* is essential in a brokering system in order to prevent malicious use for both the service components of the IBP and the end users.

2.3 TSB Messages

There are two discrete categories of messages in the proposed IBP architecture (Figure 1). The incoming messages created from Enablers that mentioned in section 2.1 and are out of the scope of this paper and the messages constructed by the TSB for internal message exchanging and routing. The messages constructed by the TSB have a common structure and consist of two parts, (a) the envelope that is constituted of four segments of fields and (b) the content that includes the initial incoming message from Enablers (section 2.1). The envelope includes the minimal information to support the required functionality so as not to increase significantly the size of the message and ensure the optimal use of the available resources. Moreover, it provides consistently the operation of TSB and its structure is presented in Figure 2.

```

<envelope>
  <originator></originator>
  <recipients>
    <recipient></recipient>
    ...
    <recipient></recipient>
  </recipients>
  <session_ID></session_ID>
  <message_ID></message_ID>
</envelope>

```

Fig. 2. The structure of envelope constructed by TSB

2.4 TSB Functionality

According to the functionalities that a brokering system has in the IBP (section 2.2), TSB is designed to be a lightweight, personalized integration solution with guaranteed reliability, which provides transparency from the application layer. In order to provide openness and flexibility, TSB should be designed following an abstract pattern and typical features [9]. Openness and flexibility allow 3rd party software developers to

integrate their services, frameworks or enablers with no or minimal code modification to the system. To meet these major challenges the TSB includes the following key functions:

- **Location Transparency:** TSB contains and configures message endpoints so that to provide message transportation. These message's endpoints are a set of interfaces (APIs [8]), that contain information about the operating capabilities both of the applications and the messaging systems, bridging them transparently and knowledge independently from the location that the requestors and the receivers have.
- **Transformation/Translation:** TSB converts messages from one format to another based on open standards (e.g. XSLT, XPath, etc) and translates them according to syntactic and semantic rules.
- **Protocol Conversion:** TSB accepts messages sent with a variety of different application layer protocols (e.g. SOAP) and converts them to a format required by the Enablers, the SR and the SB.
- **Messaging:** TSB supports synchronous, asynchronous, point-to-point, and publish-subscribe operational modes for sending and receiving messages either from the Enablers or from the SB and SR.
- **Message Routing:** TSB provides flexible and intelligent routing by the means of a dynamic router, which allows the routing logic to be modified by sending control messages. Routing is an essential feature because allows to decouple the source of the message from the ultimate destination providing transparency between message requestor and receiver.
- **Message Enhancement:** TSB checks and compares the messages before delivering them to the SB. If there is an error in the transformation phase, then retrieves the missing data based on the existing message.
- **Monitoring and Management** are mechanisms for easy monitoring the performance and controlling the runtime execution of the message flows. Moreover, provides auditing mechanisms so as to be high performing and reliable [9].
- **Security** in TSB involves authentication, authorization and encryption or decryption functionality both into incoming and outgoing messages so as to prevent malicious use and handles messages in a fully secure manner [9].

2.5 SB Functionality

The key functionality of the SB is to process a received message from TSB and interact with SR so as to find, bind and invoke services and finally integrate, orchestrate the response of the message and forward it to the TSB. SB includes the following key functions similarly to the functionalities mentioned in section 2.3:

- **Service Interaction:** SB contains interfaces (APIs) to enable interaction and communication directly with the applications and services from 3rd party providers and supports standards for web service communication (e.g. SOAP, WSDL, etc). Moreover, Java Message Service (JMS) API and the J2EE Connector Architecture (JCA) are implemented for integration between application servers and message

oriented middleware (MOM) [9]. Finally, are supported underlying protocols and communication mechanisms such as TCP, HTTP, SMTP, FTP, JBI, POP3, etc.

- **Service Integration:** The SB negotiates and enforces policies among service providers to guarantee secure service invocation.
- **Service Orchestration:** The received responses from the providers are processed using Business Process Execution Language (BPEL) and then integrated into a unified message with a common format so as to be invoked by the end user of the service.
- **Service Security:** Handles access control and authentication for messaging TSB and services provided by 3rd party providers. Moreover, encrypts and decrypts the content of messages preventing malicious interventions.
- **Service Management:** SB provides auditing facilities for monitoring the process execution and integration scenario.

3 Architecture of the Proposed IBP

In this section, according to the IBP functionality mentioned above, we analyze the architecture and Functional Entities (FEs) of the TSB and SB that compose IBP. It should be mentioned that all the Enablers and 3rd party providers are registered to the IBP so as to be widely accessible. Moreover, we consider that the monitoring, auditing and administrating mechanisms operate parallel with the TSB and SB FEs.

3.1 TSB Architectures and FEs

TSB roots the messages “onward towards to the intended recipients” by the means of the functional entities of the architecture that is presented in Figure 3.

TSB contains unique endpoints for the inbound and the outbound messages. Whenever messages either from Enablers or message responses from SB and SR trigger the TSB, the Inbound endpoint activates the Listener (step 1). Listener is permanently ‘alive’ and ready to accept messages. As soon as it receives a message it checks: (a) the validity of the message and the users’ capabilities and (b) the available resources of TSB by the means of a filter and the central data store. If the incoming message is not certified as valid, the sender of the message is properly informed about the cause of failure and the actions to be performed to restart the session.

Listener automatically forwards the message to the TSB central data store (step 2). TSB central data store contains: (a) temporarily the incoming message for security and recovery purposes until the session is released/completed, (b) temporarily information about the users’ capabilities and provide a Single-Sign-On ticket [2, 7] to be always connected, (c) rules for message translation and conversion and (d) ‘traces’ from the originated and delivered messages. The available resources of all FEs of TSB architecture are also registered in this data store. Moreover, Listener forwards the incoming message to a filter for validation (step 3). Since the message is validated and certificated, then the Message Creator (step 4) constructs the envelope for the message transition using identifying and structuring mechanisms. The enhanced

message is composed of the envelope and the originated message. This enhanced message is forwarded into a message queue (step 5). The Message Processor gets the messages from the message queue (step 6), analyzes their envelope to identify the transition / transformation / conversion conditions and implements the necessary reformatting (syntactic level) and translation (semantic level) to the payload of the message using the information that is provided by the central data store. After structuring the new message it puts it to the queue of one of the available message stores of TSB (step 7). If the structuring of the new message cannot be completed due to the lack of some information into the payload of the message, then the ‘trace’ and the prior registered information from the TSB central data store are used to re-process the message (error handling) (step 8). Additionally, the Message Allocator interacts with a filter to certify the validity of the message derived from the message stores using the information that is provided by the central data store (step 9). In the case of a failure the TSB central data store is properly updated and a new session is started by the Listener (step 10). Finally, the Outbound message is delivered to the final recipient(s) by the Router that ‘reads’ the envelope to define the end-destination(s) and if necessary, it divides or multiply the payload of the message based on the conditions of transition (step 11).

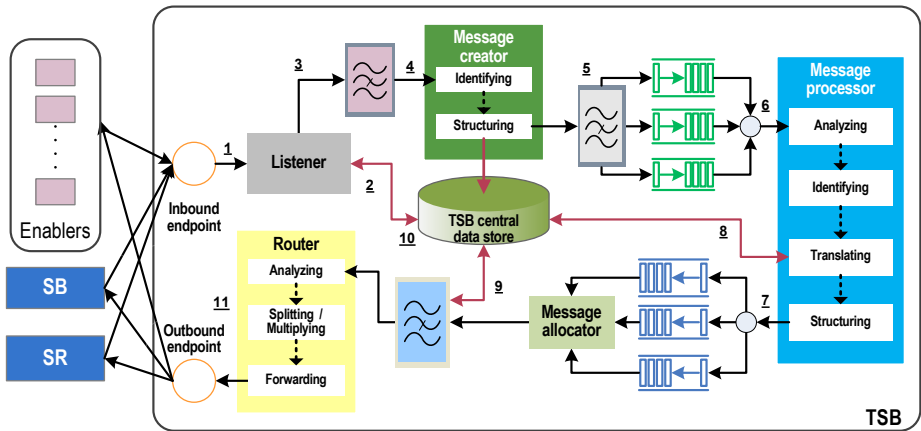


Fig. 3. TSB architecture

During the entire communication an auditing mechanism is enabled. For each process the exact time details (timestamps), successful and failed attempts are marked to monitor the performance, the effectiveness and the quality of the provided service.

3.2 SB Architectures and FEs

The architecture of SB follows the design of TSB architecture, since both of them are brokers and there are similar requirements for message administration (Figure 4). Respectively to TSB architecture, SB architecture contains a SB data store that keeps

information about the APIs that are used for the interaction with the 3rd party providers and some policy aspects so as to provide an effective communication and ensure a secure invocation of the services provided by 3rd party providers.

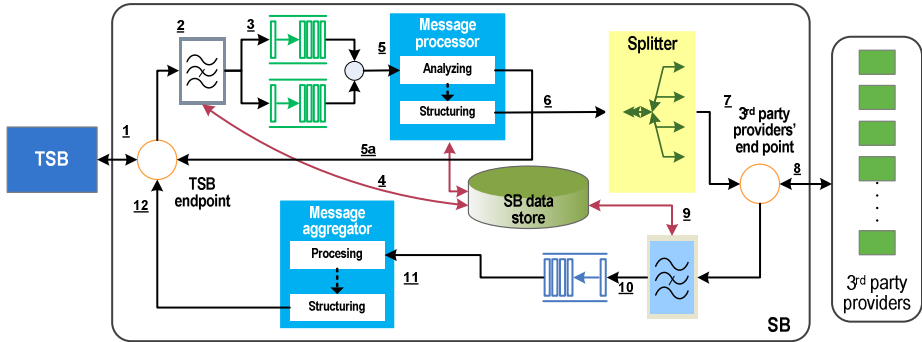


Fig. 4. SB architecture

Whenever a message triggers the TSB endpoint of SB, a session between the TSB and SB is activated (step 1). The inbound message from TSB endpoint is validated through a filter (step 2). The filter enforces some authentication and message validity control mechanisms using the information that is registered to the SB data store and puts the message to the queue of a message store (step 3). For the supervision of the entire communication a copy of the originated message is filed also to the SB data store (step 4). Then, the Message Processor gets the stored message from the queue and reads its content. The content of the message provides information about the required service of the end-user. For the efficient routing of the message into the appropriate service providers, the SB must communicate with the SR so as to find the most appropriate service for the end-user's request. For this reason, Message Processor sends back to TSB a request in order to activate/ trigger SR. SR is activated by the TSB, finds the appropriate service(s) and sends a response back to the TSB that gathers the message from SR with the service's URL and API(s). TSB forwards this message to the Message Processor following the above analyzed steps 1-5. Since Message Processor has successfully analyzed the content of the initial message, then through a structuring mechanism enhances the analyzed content with API information and the enhanced messages will continue to be delivered to the 3rd party providers (step 5). These messages are forwarded to the Splitter (step 6). In the real world, a received message is most of the times a complex process which is composed of many sub-processes for finding and binding more than one service from different service providers. The Splitter shares these messages simultaneously to the appropriate service providers so as to bind the appropriate service for the end-user (step 7). interaction with the 3rd party providers is provided through a unique 3rd party provider's endpoint (step 8). The inbound responses are filtered and checked for the validity of the messages retrieving the necessary information from the SB data store (step 9). The valid messages are put to the queue of a message store (step 10).

Moreover, The Message Aggregator processes the received messages using the BPEL and then it integrates them to a unified message (step 11), which is forwarded to TSB endpoint and finally to the TSB for continuing the session with the service applicant (step 12).

4 Implementation of the Proposed IBP

Today, in the market field there is available a variety of different ESB products such as Oracle ESB, JBOSS, OpenESB, MuleSoft ESB- MuleESB, and other [10, 11, 12, 13,]. Our IBP architecture is based on open standards and provides integration independent from the software products what will be used for implementing either the TSB or the SB. For this reason, we selected from a list of open source ESB software products [12, 13], the MuleESB [14] for a prototype implementation. The MuleESB is a very famous and not commercial product that supports a wide variety of transport protocols, data transformation, data formats, programming languages, web services, cloud connectors and security mechanisms. Moreover, according to Z. Siddiqui et al [15] analysis based on Analytical Hierarchy Process (AHP), MuleESB is more preferable, information secure, high available and interoperable in contrast to FuseESB. In contrast with the JBoss [16], MuleESB's performance in a typical message routing without business logic was 3 times faster.

The proposed IBP has been implemented as prototype to provide medical personalized services and its pilot operation performed into the laboratory. For the exchanged messages the HL7 v2.x and HL7 v3 standards were applied and for the implementation of them the NeoTool Library was used. The engines of IBP that provide the messaging, the routing and the translation / conversion mechanisms were implemented as independent Java modules using the Eclipse IDE that is a platform compatible with the MuleESB.

For evaluating the prototype implementation a simple communication scenario was built utilizing already developed Medical Information Systems (MISs). In this scenario, we consider that the tele-healthcare service used by the physician is authorized to receive data from a MIS provided that the two systems ought to satisfy and fulfill the appropriate policy agreements and Service Level Agreements (SLAs). In this case study, we also consider that the doctor's application is a telemedicine service using the HL7 v.2.0 for the message administration and transformation. In the same time the MIS is an EHR system that applies the HL7 v.3.0 standard for the organization, administration and transmission of messages.

In the pilot operation, the IBP platform is triggered by the physician's application/enabler to retrieve data (e.g. user's profile and laboratory exams) stored in an EHR (of the 'healthcare center' where the patient has been hospitalized). The IBP receives a message/ request for data retrieval, the Listener of TSB is automatically activated and the whole service recovery process in collaboration to the SB (analyzed in section 3) takes place. The SB authorizes, authenticates and validates doctor's access for this request. Since SB identifies the physician's privileges, then sends via the IBP a request to trigger the appropriate EHR's units, retrieves, collects and sends the requested on demand data to the physician.

To monitor the whole process and evaluate the response time of the IBP, a logging service (auditing service) was implemented, which is activated during the entire communication and messages transformation / exchange creating log files with all the necessary information regarding: successful or non - execution of the message processing and transmission from one system to the other, the response times of IBP and the overall time of the session (from receiving the original message to return the requested data).

These parameters were the base for the extraction of statistical and evaluation of quality factors for the developed IBP, according to the ISO 9126 model's characteristics [17] that regard the Functionality, the Reliability and the Efficiency. The results for these three factors are presented in the following diagrams and the Table includes the relative logged measures.

Table 1. Logged evaluation parameters of IBP

Evaluation parameters	Measurements	Statistical	
<i>Set 1: Number of transferred messages</i>	<i>(messages)</i>		
Messages originally sent	616	-	-
Delivered & acknowledged messages	606	98.38%	1.62%
Rejected by filters	7	1.14%	98.86%
Resent (delivered / not)	134	99.26%	0.74%
<i>Set 2: Times and delays</i>	<i>(sec)</i>		
Memory queuing time	1.7	-	-
Resending delay	2.1	-	-
Total transferring time	24	-	-
Total completing session time	48	-	-
<i>Set 3: Sessions through IBP</i>	<i>(sessions)</i>		
Attempted sessions	388	-	-
Complete TSB-SB sessions	383	98.71%	1.29%
Complete MIS-IBP sessions	381	98.20%	1.80%
Complete transfers of messages	380	97.94%	2.06%

By analyzing the results of this process, it came out that the IBP achieves to satisfy this specific and simple communication by finalizing successfully all the processes for message transformation and elaboration, even though the response times remain in tolerated, but not in satisfactory levels. This arises the need for optimization the modules of IBP that implement the message processing concerning the algorithms that they use and the lack of parallel processors that share the message traffic.

From the whole process of implementation and the pilot operation of the IBP, it was validated that for the communication of the external systems no alteration is needed as regards the end-user services that these provide.

Meaning that, the architecture and the communication models provided and supported by the IBP do not interfere at all to the harmonious operation of the existing systems and no effort is needed for the integration of IBP to already applied business models into a healthcare organization.

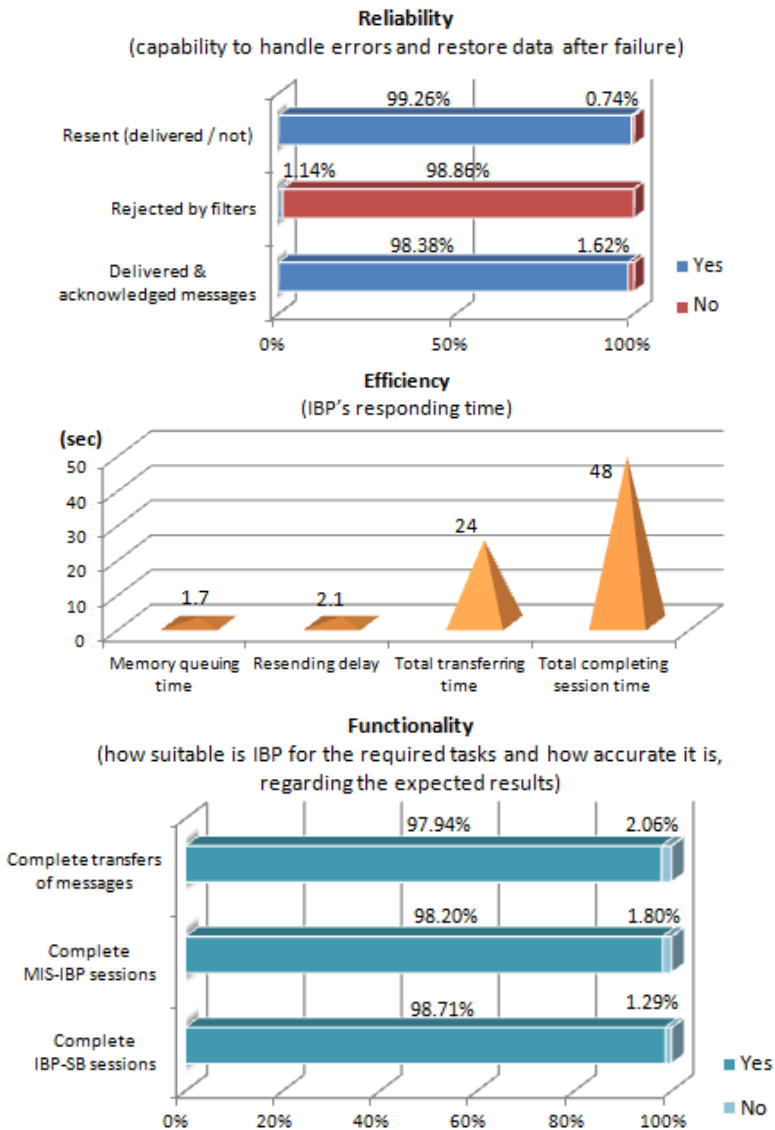


Fig. 5. IBP quality factors

5 Conclusion

This paper proposes an open IBP architecture for the eHealth domain. We analyzed thoroughly the architecture and the functionality of each of the core brokering components (TSB and SB) for message and service delivery purposes. In order to clarify the potential role of the IBP architecture in the real world for personalized and mobile services, we described analytically the message and service delivery processes that provide location transparency and are aware of contexts. Finally a prototype was implemented using an open source, non commercial software product, the MuleESB.

For future work, we plan to study the performance of the IBP, evaluating the IBP's behavior in real world. This real world IBP environment will be consisted of more than four autonomous and independent brokers such as context broker that will enhance the functionality of IBP with advanced personalized features.

References

1. Continua Health Alliance, <http://www.continuaalliance.org/index.html>
2. Andriopoulou, F., Lymberopoulos, D.: A new platform for delivery interoperable Telemedicine services. In: Proc. Second Int. ICST Conf on Wireless Mobile Communication and Healthcare, Kos Island, October 5-7, pp. 181–188 (2011)
3. Gonzalez, L., Llambias, G., Pazos, P.: Towards an e-health integration platform to support social security services. In: Proc. 6th International Policy and Research Conference on Social Security, Luxembourg, September 29-October 1(2010)
4. Van Hoecke, S., Steurbaut, K., Taveirne, K., De Turck, F., Dhoedt, B.: Design and implementation of a secure and user-friendly broker platform supporting the end –to-end provisioning of e-homecare services. *Journal of Telemedicine and Telecare* 16, 42–47 (2010)
5. Open eHealth Foundation, <http://www.openehealth.org/display/OEHF/Foundation>
6. The Apache Software Foundation, <http://www.apache.org/>
7. Andriopoulou, F., Lazarou, N., Lymperopoulos, D.: A proposed Next Generation Service Delivery Platform (NG-SDP) for eHealth domain. In: Proc. of 34th Annual Intern. Conf. of the IEEE Engineering in Medicine and Biology Society, San Diego, August 28-September 1 (2012)
8. ETSI ES 203 915-3 V1.2.1, Open Service Access (OSA); Application Programming Interface (API); Part 3, Framework, Parlay 5 (2007)
9. Menge, F.: Enterprise Service Bus. In: Proc. of Free and Open Source Software Conference (2007)
10. Woolley, R.: Enterprise Service Bus (ESB) Product Evaluation Comparisons. UTAH Department of Technology Services (October 18, 2006)
11. Mr. Gupta, Enterprise Service Bus Capabilities Comparison, in Project Performance Corporation Part of AEA Group (April 2008), <http://www.ppc.com/documents/enterprisebus.pdf>
12. Pronschinske, M.: Top Open Source ESB Projects (October 2009), <http://architects.dzone.com/news/top-open-source-esbs>
13. Cope, R.: Comparison of Open Source ESB Solutions (August 2010), <http://www.openlogic.com/>

14. MuleSoft™, <http://www.mulesoft.com/company>
15. Siddiqui, Z., Abdullah, A.H., Khan, M.K., Alghathbar, K.: Analysis of enterprise service buses based on information security, interoperability and high availability using Analytical Hierarchy Process (AHP) method. *Inter. Journal of Physical Sciences* 6(1), 35–42 (2011)
16. Tiwari, N.: JBOSS ESB vs Mule Performance (August 2008), <https://community.jboss.org/message/506587>
17. ISO/IEC 9126, International Standard, Information Technology – Software Product Evaluation – Quality characteristics and guidelines for their use (1991), http://www.usabilitynet.org/tools/r_international.htm