Enabling Tenant-Aware Administration and Management for JBI Environments

Steve Strauch*, Vasilios Andrikopoulos*, Santiago Gómez Sáez*, Frank Leymann*, Dominik Muhler‡

* Institute of Architecture of Application Systems, University of Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

‡ Center of Excellence F&R, SAP (Schweiz) AG, Switzerland
dominik.muhler@sap.com

BIBTEX:

```
@inproceedings{StrauchALM2012,
  author    = {Steve Strauch, Vasilios Andrikopoulos, Santiago Gómez Sáez,
               Frank Leymann, and Dominik Muhler},
  title     = {Enabling Tenant-Aware Administration and Management for JBI
               Environments},
  booktitle = {Proceedings of the 5th IEEE International Conference on Service-
               Oriented Computing and Applications, SOCA 2012,
               17-19 December 2012, Taipei, Taiwan},
  year      = {2012},
  pages     = {206-213},
  publisher = {IEEE Computer Society}
}
```

Enabling Tenant-Aware Administration and Management for JBI Environments

Steve Strauch, Vasilios Andrikopoulos,
Santiago Gómez Sáez, Frank Leymann
Institute of Architecture of Application Systems (IAAS),
University of Stuttgart, Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

Dominik Muhler
Center of Excellence F&R, SAP (Schweiz) AG, Switzerland
dominik.muhler@sap.com

***Abstract*—Enterprise Service Buses (ESBs) constitute a core middleware technology for each modern Service-Oriented Architecture (SOA) solution. Given the popularity of the Cloud paradigm, which is based on fundamental SOA concepts, it is only therefore natural to look into how ESBs can be transformed into native building blocks for Cloud platforms. As a first step of this effort, in this work we investigate how ESBs can become multi-tenant aware, i.e. able to support multiple tenants and their users sharing the same ESB instance. A generalized architecture based on the JBI specification implemented by a number of open source ESBs is presented for this purpose. We demonstrate the feasibility of our proposal by means of a proof of concept realization and we evaluate the performance of our solution against a non multi-tenant ESB that was used as the baseline for our implementation.**

***Keywords*-Multi-tenancy; Enterprise Service Bus (ESB); JBI specification; Platform as a Service**

I. Introduction

The Enterprise Service Bus (ESB) technology [1] as the messaging hub between applications addresses the fundamental need for application integration and as such, in the last years it has become ubiquitous in service-oriented enterprise computing environments. ESBs control the message handling during service invocations and are at the core of each Service-Oriented Architecture (SOA) [2]. Given the increasing interest of the industrial and research community in Cloud computing [3], and the fact that the Cloud computing paradigm is discussed in terms of the creation, delivery, and consumption of services [4], it is therefore essential to investigate into how the ESB technology can be used efficiently in a Cloud-oriented environment.

For this purpose, in this work we focus on making ESBs *multi-tenant aware*. Multi-tenancy and virtualization are the key enablers that allow Cloud computing solutions to serve multiple customers from a single system instance. Using these techniques, Cloud service providers maximize the utilization of their infrastructure, and therefore increase their return on infrastructure investment, while reducing the costs of servicing each customer. While many industrial strength solutions exist for virtualization[1], multi-tenancy is an issue still under research and as such, it is the goal for this work. Multi-tenancy has been defined in different ways in the literature for SOA and middleware, see for example [5], [6], [7], [8]. Such definitions however do not address the whole technological stack behind the different Cloud service models as defined in [3] (i. e. IaaS — Infrastructure as a Service, PaaS — Platform as a Service, SaaS — Software as a Service). In this work we define multi-tenancy as *the sharing of the whole technological stack (hardware, operating system, middleware, and application instances) at the same time by different tenants and their corresponding users.*

We distinguish between two different consumer types for multi-tenancy purposes: *tenants* and *users*. Tenants separate the consumers using a multi-tenant service or application into groups like companies, organizations or departments. These groups are not necessarily completely disjoint since a consumer may belong to more than one tenant at the same time. Users enable the differentiation between consumers potentially belonging to more than one tenant and therefore introduce a finer level of granularity. Consider for example the case of a provider offering a multi-tenant Cloud service in the taxi domain managing client requests and assigning them to drivers. A small taxi company using this service is a tenant of this multi-tenant service. The customers of the taxi company are consuming the service and therefore act as the users of the tenant. Such differentiation between consumers using users is essential for performing tasks like accounting and billing on behalf of the tenant.

In this context, making an ESB multi-tenant aware means that *the ESB is able to manage and identify multiple tenants and their users, providing tenant-based identification and hierarchical access control to them.* In other words, the ESB should provide the appropriate mechanisms that allow tenant applications to seamlessly interact with it while sharing one (logical) instance of the ESB. Given the role of the ESB middleware technology in the technological stack, there are two fundamental aspects of multi-tenancy awareness: communication (i.e. supporting message exchanges isolated per tenant), and administration and management (i.e. allowing each tenant to configure and manage individually their communication endpoints at the ESB). Isolation can be further

[1]See for example: http://www.xen.org, http://www.vmware.com/products/, and http://www.flexiant.com/

decomposed into *data* and *performance* isolation between tenants of the same system. In this paper we scope the discussion to the administration and management aspect of multi-tenant aware ESBs.

More specifically, in the following sections we present a framework enabling multi-tenant aware administration and management of Java Business Integration (JBI) environments. The JBI specification [9], created by the Java Community Process, defines a Java framework that standardizes the interoperation between service containers, connectivity services, and integration services. A number of middleware technologies like ESBs and application servers implement the JBI specification, e.g. the open source solutions Open ESB[2], Petals ESB[3], Apache ServiceMix[4], and GlassFish[5]. By basing our approach on the JBI specification we therefore ensure that we produce a general and reusable solution that can be replicated across different ESB solutions (and other technologies that implement the JBI specification). Our contribution therefore can be summarized by offering:

- A framework enabling the multi-tenant aware administration and management of JBI environments.
- A proof-of-concept implementation of our proposal based on the open source ESB Apache ServiceMix.
- A performance evaluation of our implementation, compared against the Apache ServiceMix version that we used as a baseline for the development of our solution.

The remaining of the paper is structured as follows: Section II briefly summarizes the JBI specification and presents our proposal for an architectural framework enabling multi-tenant administration and management in JBI environments, by first identifying the requirements for this purpose. Section III discusses the realization of this framework using Apache ServiceMix as a proof-of-concept for our proposal and Section IV provides a performance evaluation for this realization. The paper closes with Section V and Section VI summarizing related work, and concluding with some future work, respectively.

II. Multi-Tenant Aware Administration and Management of a JBI Architecture

In this section we briefly introduce the JBI specification before we define the functional and non-functional requirements for enabling multi-tenant aware administration and management of JBI environments. The requirements have been identified during our work in the EU research project 4CaaSt[6], collaborations with industry partners, and through literature review. Afterwards we propose an architecture satisfying these requirements.

[2]Open ESB: http://openesb-dev.org

[3]Petals ESB: http://petals.ow2.org

[4]Apache ServiceMix: http://servicemix.apache.org

[5]GlassFish: http://glassfish.java.net

[6]The 4CaaSt project: http://www.4caast.eu

A. Java Business Integration Environment

The JBI specification by the Java Community Process creates a standard-based environment for integration solutions and specifies the interaction of *JBI components* installed into a *JBI container* [9]. A JBI container facilitates plugging-in JBI-compliant components interacting through a *Normalized Message Router* (NMR). The NMR is a message-oriented mediator and ensures loose coupling between JBI components. JBI components consume or provide services and describe them according to the WSDL 2.0 specification [10]. There are two different types of JBI components: *Binding Components* (BCs) and *Service Engines* (SEs). BCs provide connectivity to external services and mediate between external protocols and the NMR. SEs provide business logic and transformation services inside the JBI container. Additionally, JBI specifies a management framework based on the Java Management Extensions (JMX) specification for the configuration of JBI components to cope with individual integration tasks. The management framework allows to install JBI components, to deploy service artifacts, called *service units*, to configure them, control their state, and control the overall state of the JBI container. Different service units can be packaged as *service assemblies*.

B. Functional and Non-Functional Requirements

Following the discussion about multi-tenancy of PaaS components, in the context of project 4CaaSt we identified and categorized a set of *functional* and *non-functional* requirements for multi-tenant aware administration and management. Toward this goal we also refined the multi-tenancy characteristics (e.g. tenant awareness) identified already in the literature, e.g. [11], [5], [6], [7], [8].

Functional requirements: The following functionalities must be offered by any JBI environment providing multi-tenant aware administration and management.

FR$_1$ *Tenant awareness*: A JBI environment must be able to manage and identify multiple tenants, i.e. tenant-based identification and hierarchical access control for tenants and their users must be supported.

FR$_2$ *Tenant-based configuration*: The installation of JBI components and the deployment of corresponding configurations for a certain tenant should be managed in a transparent manner by the JBI environment.

FR$_3$ *Tenant-specific interfaces*: A set of customizable interfaces must be provided, enabling administration and management of tenants and users, including both GUIs and Web services interfaces.

FR$_4$ *Shared registries*: As the JBI environment will be embedded in a PaaS platform with other applications demanding similar information, the approach must come with a shared for other PaaS components registry of tenants/users and a shared registry of configurations.

FR_5 *Backward compatibility*: The JBI environment should be able to handle non multi-tenant aware administration and configuration by enabling installation of JBI components and deployment of service artifacts that are not multi-tenant aware.

Non-functional requirements: In addition to the required functionalities, JBI environments should also respect the following properties.

NFR_1 *Tenant Isolation*: Tenants must be isolated to prevent them from gaining access to other tenant's data (i.e., *data isolation*) and computing resources (i.e., *performance isolation*). Data isolation can be further decomposed into *communication isolation*, referring to keeping the message exchanges for each tenant separate, and *application isolation*, referring to preventing applications and services of one tenant from accessing data of another tenant's applications or services.

NFR_2 *Security*: The necessary authorization, authentication, integrity, and confidentiality mechanisms must consider and enforce tenant- and user-wide security policies when required.

NFR_3 *Reusability & extensibility*: The multi-tenancy enabling mechanisms and underlying concepts should not be solution-specific and depend on specific technologies to be implemented. JBI containers and components should therefore be extensible when required and reusable by other components in the PaaS model (as for example in the case of the shared registries functional requirement).

NFR_4 *Tenant data consistency*: Distributed transactions implementing the ACID principle have to ensure data integrity between different components modifying data specific to each tenant, e.g. configuration information.

C. Multi-tenant Aware Administration and Management Framework

Figure 1 provides an overview of our proposal for a generic multi-tenant aware administration and management framework, which fulfills the requirements identified in the previous section. More specifically, the three layer architecture consists of a *Presentation* layer, a *Business Logic* layer, and a *Resources* layer. In the following we present the components required for each layer of the architecture and how they address the functional and non-functional requirements described in Section II-B in a bottom-up fashion.

1) Resources layer: The Resources layer consists of a *JBI Container Instance Cluster* and a set of registries. The JBI Container Instance Cluster bundles together multiple *JBI containers*. In the simplest case, the JBI Container Instance Cluster may consist of only one (running) JBI container handling all tenants and users using an implementation of the framework for multi-tenant aware administration and management. Since this however may create performance issues, a clustering mechanism similar to the one provided for example by Apache ServiceMix is recommended to be used. In order to enable multi-tenancy in any JBI Container Instance, these containers, and all other JBI components installed in the container (see Section II-A), must be *multi-tenant aware*, i.e. able to operate with multiple tenants and users administrating and configuring the same instance of the JBI container. For a JBI Container Instance in particular, this means that BCs and SEs are able to handle service assemblies containing tenant and user specific configuration information, and process them accordingly in a multi-tenant manner (FR_2). For example, a new tenant and user specific endpoint has to be created for interaction with a JBI component whenever a service artifact is deployed to this JBI component. Thereby, communication isolation is ensured (NFR_1).

The installation/uninstallation and configuration of BCs and SEs in a JBI Container Instance is performed by means of a set of *standardized interfaces*. While these interfaces, and all other components of the JBI Container Instance, are multi-tenant aware, special care has to be taken to ensure backward compatibility (FR_5). This means that installation and configuration of non multi-tenant aware JBI components must still be possible. Processing of non multi-tenant aware service assemblies has to be performed normally. Within the Resource layer we also introduce three different types of registries. The *Service Registry* stores the services registered with the JBI environment, as well as the service assemblies for the configuration of the BCs and SEs installed in each JBI Container Instance in the JBI Container Instance Cluster in a tenant-isolated manner [12] (NFR_1). Currently we are focusing on the approach that each JBI Container Instance of the JBI Container Instance Cluster has the same BCs and SEs installed. As the BCs and SEs are common, and in order to offer the possibility of horizontal scalability support [13], a

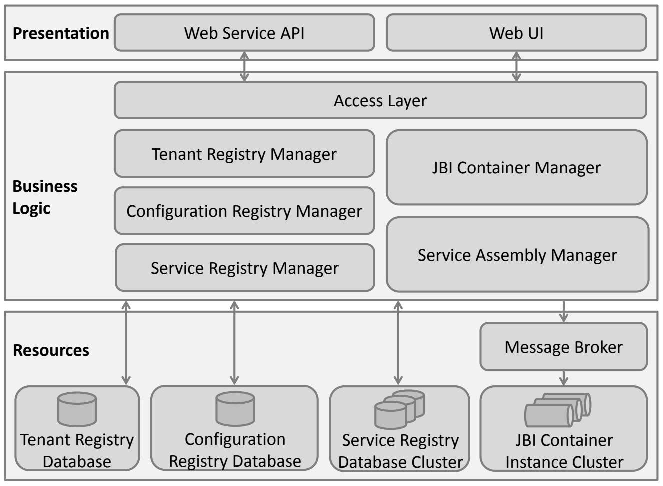

Figure 1. Overview of multi-tenant aware Administration and Management Framework

load balancer (not shown in Fig. 1) must retrieve the required service assemblies from the Service Registry and deploy them to the corresponding JBI components after installing the JBI components when starting an additional JBI Container instance, e.g., to cover increased load. As we propose to share the Service Registry with other PaaS components, e.g., composition engines (FR_4), and for the sake of reusability (NFR_3), we recommend to realize the Service Registry as a database cluster to avoid performance bottlenecks.

The *Tenant Registry* stores a set of users for each tenant and the corresponding unique identifiers (FR_1). Additionally, each tenant and user may have associated properties such as tenant or user name represented as key-value pairs (NFR_3). Moreover the password required for login before administration and configuration of the JBI environment is stored in the Tenant Registry represented as hash value generated with MD5 (NFR_2). All these data are stored in a in a multi-tenant manner (NFR_1). The *Configuration Registry* stores all configuration data created by a tenant and the corresponding users, except from the service registrations and configurations stored in the Service Registry. The Configuration Registry stores for example the configuration of JBI Container Instances (FR_2), the mapping of JBI Container Instances to tenants (FR_1), and the mapping of permissions to roles according to the role-based access control mechanisms offered by the Access Layer component in the next layer (NFR_1 and NFR_2).

When a tenant or user interacts with the multi-tenant aware JBI environment, the data in more than one registry might have to be changed. Consequently all operations and modifications on the underlying resources have to be handled as distributed transactions based on a two-phase commit protocol [14] so that a consistent state of all resources is ensured (NFR_4). As many JBI components from several JBI Container Instances in the cluster might participate in the distributed transaction and this might lead to performance bottlenecks we recommend to decouple them from the distributed transactions using messaging with guaranteed delivery [15], e.g. a Message Broker (see Fig. 1).

2) Business Logic layer: The Business Logic layer contains an *Access Layer* component, encapsulating the functionality that ensures tenant awareness and security (FR_1 and NFR_2, respectively). The Access Layer acts as a multi-tenancy enablement layer [5] based on role-based access control [16]. The tenants and their corresponding users have to be identified and authenticated once when the interaction with the JBI environment is initiated. Afterwards, the authorized access should be managed by the Access Layer transparently. Prior to authentication and identification of tenants and users, the Access Layer component handles authorization by registering tenants and users and granting them access to JBI Container Instances (NFR_2). Therefore, in case of a multi-tenant aware interaction with the system, each tenant and user has to identify themselves by providing a unique *tenantID* and *userID* (FR_1).

In addition to the Access Layer component, the Business Logic layer also contains a set of *Managers* (Fig. 1) encapsulating the functionality to interact with underlying components in the Resources layer. The *Tenant Registry*, *Configuration Registry*, and *Service Registry Managers* implement the business logic required to retrieve and store data in the corresponding registries. The *JBI Container Manager* installs and uninstalls BCs and SEs in each JBI Container in the cluster, while the *Service Assembly Manager* takes care of configuring them appropriately by deploying and undeploying service artifacts. Both managers are using the standardized interfaces provided by each JBI Container Instance for this purpose, as discussed in the Resources layer.

3) Presentation layer: The Presentation layer contains two components allowing the customization, administration, and interaction with an implementation of the multi-tenant aware administration and management JBI environment: the *Web UI* and the *Web service API*. The Web UI offers a customizable interface for human and application interaction with the system, allowing for the administration and management of tenants and users (FR_3). The Web service API offers the same functionality as the Web UI, but also enables the integration and communication of external components and applications (NFR_3). For both interface mechanisms, security aspects such as integrity and confidentiality of incoming messages must be ensured (NFR_2) by, for example, using Web Services Security (WS-Security) for the Web service API and Secure HTTP connections for the WebUI. A discussion on the particular mechanisms to be used for this purpose is out of scope of this work.

III. Realization

For purposes of implementation of our approach we extended the open source ESB Apache ServiceMix version 4.3.0 (hereafter referred to simply as ServiceMix) and integrated it with the newly developed Web application *JBIMulti2* for enabling multi-tenant aware administration and management. The name *JBIMulti2* is based on the fact that the Web application enables multi-tenant aware administration and management of both BCs and SEs. The extension to ServiceMix and the integration with the JBIMulti2 is illustrated in the form of a deployment diagram in Fig. 2. Components and libraries being reused are marked in gray. ServiceMix is based on the OSGi Framework [17]. OSGi bundles realize the ESB functionality complying to the JBI specification [9]. In the following we present the realization of the components in the different layers of the framework in a bottom-up fashion, similarly to the presentation of Section II-C.

A. Resources layer

ServiceMix is provided with several JBI components, i.e. the SE for Apache Camel enables usage of Enterprise Integra-

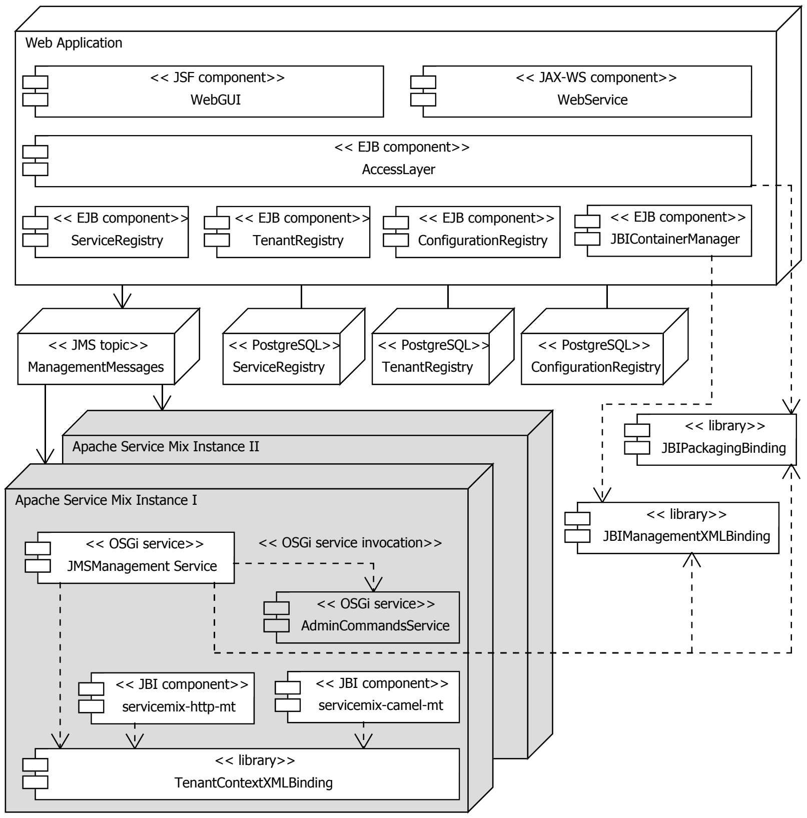

Figure 2. Deployment diagram of prototype realization of multi-tenant aware Administration and Management Framework

tion Patterns [15]. The original ServiceMix BC for HTTP and the original Apache Camel SE were extended in our prototype in order to support multi-tenancy (see the *servicemix-http-mt* and *servicemix-caml-mt* components in Fig. 2). In addition, ServiceMix was extended by an OSGi-based management service (*JMSManagement Service* component), which listens to a JMS topic for incoming management messages sent by the Web Application (Fig. 2). For JMS messaging we use the message broker Apache ActiveMQ version 5.3.1[7]. The *ServiceRegistry*, *TenantRegistry*, and *ConfigurationRegistry* components are realized based on PostgreSQL version 9.1.1[8].

B. Business Logic layer

As the Web Application might modify more then one resources, all operations are handled within distributed transactions. The Web Application itself implements the Presentation and Business Logic layers of the proposed architecture (Fig. 2). It is running in the Java EE 5 application server JOnAS version 5.2.2[9], which can manage distributed transactions. As the management components of the underlying resources are implemented as EJB components, we use container-managed transaction demarcation, which allows the definition of transaction attributes for whole business methods, including all resource changes [18].

As the AccessLayer acts as a multi-tenancy enablement layer based on role-based access control we defined two distinct categories of roles interacting with the system:

- The *system role* differentiates between *system administrators* and *tenant users*, with each tenant user belonging to one tenant. Both system roles are mutually exclusive.
- The *tenant role* classifies tenant users into *tenant administrators* and *tenant operators*.

As the system administrator configures the whole system and assigns quotas of resource usage he does not belong to any tenant. Thus, the system administrator has unlimited permissions and is allowed to interfere in the actions of the tenant users. The tenant users consume the quotas of resource usage to deploy service assemblies or to register services. Tenant administrators define roles and assign permissions to them. These roles are assigned to tenant operators who then access the resources using a resource contingent given by the tenant administrator. Each tenant user can have multiple tenant administrator roles and tenant operator roles. The union of the permissions of all tenant roles determines which tasks a tenant user can execute. A contingent defines a group of resources of the same type such as a concrete BC, SE, or WSDL service description and the maximum number of

[7]Apache ActiveMQ: http://activemq.apache.org
[8]PostgreSQL: http://www.postgresql.org
[9]Java Open Application Server: http://wiki.jonas.ow2.org

resources the group can contain. It is important that the system administrator assigns a default tenant administrator role to at least one tenant user to enable the corresponding tenant to perform actions. This scheme realizes the Role-Based Access Control model [16].

C. Presentation layer

The Presentation layer consists of the *WebGUI* and the *Web Service API* (Fig. 2). The *WebGUI* has been specified and designed based on JavaServer Faces version 1.2, but the implementation is still ongoing. Users of one system role have a completely separated WebGUI from users of another system role.

The Web Service API of the Web Application is realized based on the Java API for XML-Based Web Services version 2.0. JBIMulti2 Web service interface has to ensure integrity, confidentiality, and authentication of incoming messages. As the realization complies to the WS-Security specification, SOAP messages sent to the Web service API can be signed and encrypted. Security tokens contain keys or other data that allow the receiver to validate signatures of message parts or decrypt message parts. The Web service is based on the WS-Security X.509 certificate token profile, providing asymmetric encryption using key pairs. As we conceive confidentiality and integrity on a per tenant basis it is not required that tenant users own a X.509 certificate. On the one hand each tenant interacting with the Web service API has to be aware of the public key of JBIMulti2. On the other hand JBIMulti2 has to know the public keys of all tenants, stored as X.509 certificates in the Tenant Registry, but not validated at a certificates authority. Authentication is implemented by using a custom SOAP header element named TenantContext, which is encrypted and signed. The Tenant Context contains a tenantID and userID both represented as UUIDs, and the password of the user. The WSDL documents of the Web service API are available at http://tiny.cc/web-service-wsdls.

All artifacts required to install and setup JBIMulti2 including a manual are publicly available at http://tiny.cc/JBIMulti2-install.

IV. Evaluation

For purposes of evaluating the performance of our extension of Apache ServiceMix we used the Direct Proxy Service scenario of the Performance Test Framework for ESB Implementations[10]. This scenario aims to demonstrate the ability of an ESB to act as a virtualization layer for backend Web services, operating as a proxy between a client (the benchmark driver) and a simple echo service on the provider side. Following the test parameters set by the benchmark, we used messages with 1K payload sent by 20, 40, 160, 640 and 1280 concurrent users. These users were equally divided among 1, 2, 4 and 10 tenants, creating 4 test cases: MT/1T, MT/2T, MT/4T and MT/10T, respectively. For each of these cases, one request message of 1Kb payload for each user (composed of random characters) was sent by the driver to the ESB. The total time in receiving the receipt acknowledgment by the echo service for each message was measured, allowing us to calculate the average response time and throughput at the ESB. For comparison purposes, we measured the same test cases for the baseline (non multi-tenant) implementation of Apache ServiceMix; instead of tenants however, in this case we provided endpoints of the same service implementation (NonMT/1E, NonMT/2E, NonMT/4E and NonMT/10E). A warm-up phase of 400 messages for each endpoint or tenant was used before we took measurements.

[10]Performance Test Framework http://esbperformance.org

The test cases were run using Flexiant's Flexiscale offering[11] and two Virtual Machines – VM1 with 6GB RAM and 3 CPUs, and VM2 with 4GB RAM and 2 CPUs. Both VM1 and VM2 run the Ubuntu 10.04 OS. In the VM1 the extended Apache ServiceMix is deployed, which required also the deployment of the following components: PostgreSQL 9.1.1 database, Jonas 5.2.2 server and Tomcat 7.0.23 server. The endpoints deployed in ServiceMix are using HTTP-SOAP. These were deployed directly in the ServiceMix deployment folder (for the non multi-tenant scenarios) and through the JBIMulti2 Application (for the multi-tenant cases). In VM2, an Apache Tomcat 7.0.23 instance was deployed with the echo Web service, the benchmark driver, and Wireshark 1.2.7 for monitoring HTTP requests and responses.

Figures 3 and 4 illustrate the average response time and throughput, respectively, for the different test cases. Figure 3 shows a significant performance deterioration when comparing the baseline ESB implementation with 1 service endpoint against our implementation with 1 tenant. This deterioration however is reduced dramatically as the number of endpoints/tenants increase, resulting in very similar performance results when comparing 10 endpoints with 10 tenants, and for the same amount of requests. A similar

[11]Flexiscale http://www.flexiscale.com/

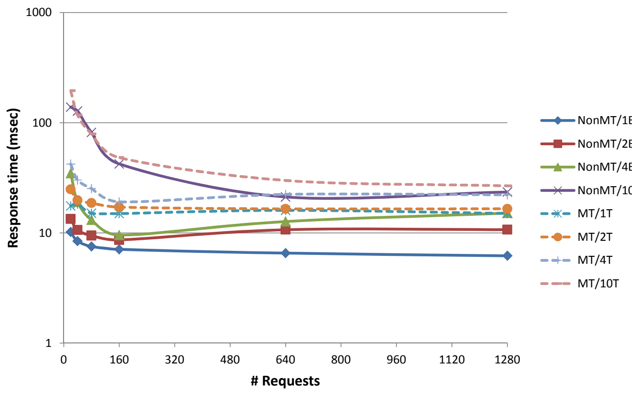

Figure 3. Average response time per case (log/normal)

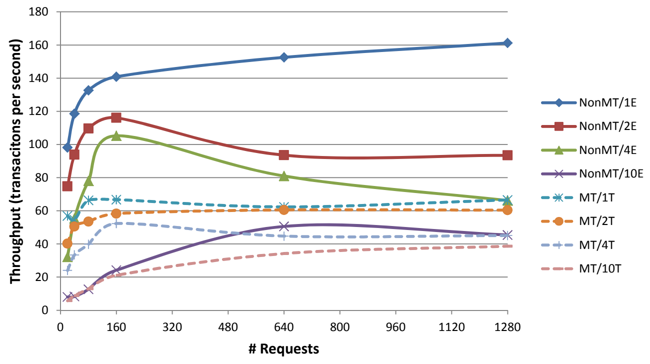

Figure 4. Average throughput per case

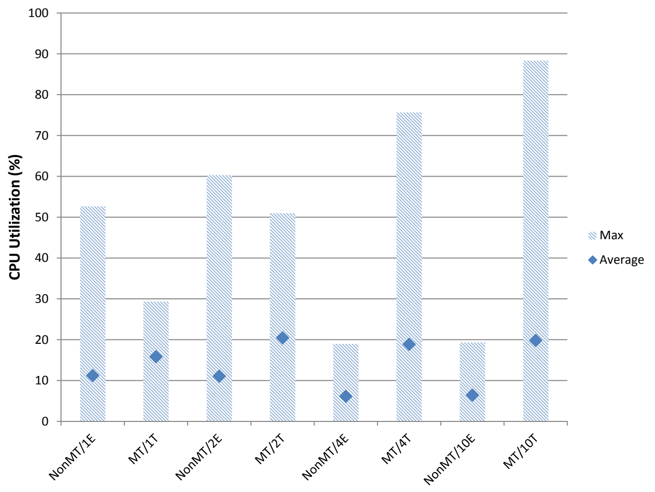

Figure 5. Maximum and average CPU utilization per case (normalized over all CPUs)

conclusion can be drawn for the amount of transactions per second: as the number of endpoints (for the non multi-tenant cases) and tenants (for the multi-tenant cases) increases, the throughput seems to be converging. Given the fact that ESBs are usually expected to operate with thousands of concurrent users these results are encouraging; however it is true that there is much space for improvement and optimization in our implementation.

Figure 5 shows the maximum and average CPU utilization for both the baseline and the extended ESB implementations. The utilization was measured throughout the execution of all the requests in each test case, and was normalized for the number of processors in VM1. Figure 5 shows that the average CPU utilization is consistently higher for the multi-tenant implementation and for the same amount of requests; maximum CPU utilization however varies. When combined with the results of the previous figures, it can be concluded that our multi-tenant ESB implementation increases CPU utilization for levels of performance that are comparable to the non multi-tenant one. While each service tenant however can be configured and managed individually, as discussed in the previous sections, each endpoint in the non multi-tenant implementation can only be configured once and uniformly across all consumers of the service.

Further evaluation of the performance of our proposal is in any case required, including for example the memory needs of JBIMulti2 and the Extended Apache ServiceMix, and investigating the trends in CPU (and memory) utilization in depth. In addition, a comparison of the performance of our multi-tenant solution against a) using a separate instance per cluster for each tenant, and b) horizontally scaling the ESB by adding more VM images and splitting the traffic between them is required. This is a direction that we are currently working on.

V. Related Work

Existing approaches on enabling multi-tenancy for middleware typically focus on different types of isolation in multi-tenant applications for the SaaS delivery model, see for example [5]. As discussed also in [8] however, only few PaaS solutions offer multi-tenancy awareness allowing for the development of multi-tenant applications on top of them. The work of Walraven et al. [8] follows a similar approach to ours with respect to using tenant context information to allow for multi-tenant aware administration and management, while ensuring data isolation. Our work however proposes a more generic approach built around any ESB technology that complies with the JBI specification, and does not require the implementation of a dedicated support layer for these purposes.

Focusing on the ESB technology, in [19] we surveyed a number of existing ESB solutions (ServiceMix, Microsoft BizTalk Server, JBoss ESB, Mule ESB, OW2 Petals ESB, IBM WebSphere ESB, and WSO2 ESB) and evaluated their multi-tenancy readiness. Our investigation showed that the surveyed ESB solutions in general lack in support of multi-tenancy. Even in the case of products like IBM WebSphere ESB[12] and WSO2 ESB[13] where multi-tenancy is part of their offerings, multi-tenancy support is implemented either based on proprietary technologies like the Tivoli Access Manager (in the former case), or by mitigating the tenant communication and administration on the level of the message container (Apache Axis 2 [14] in the latter case). In either case, the used method can not be applied to other ESB solutions and as a result no direct comparison of the applied multi-tenancy enabling mechanisms can be performed.

The approach presented in this paper differs from existing approaches by integrating multi-tenancy independently from the implementation into the ESB. Therefore, our solution can also be applied and reused to enable multi-tenancy for other PaaS offerings, e.g., composition engines. Moreover our architecture realizes a broader range of functional and

[12]IBM WebSphere ESB: http://tiny.cc/IBMWebSphereESB
[13]WSO2 ESB: http://wso2.com/products/enterprise-service-bus/
[14]Apache Axis: http://axis.apache.org/axis2/java/core/

non-functional multi-tenancy requirements, in contrast to the existing approaches focusing on a subset of them.

VI. Conclusions and Future Work

In the previous sections, we proposed an approach for making ESB solutions that comply to the JBI specification multi-tenant aware, i.e. able to serve multiple consumers from a single system instance. We identify two aspects for multi-tenancy awareness: communication, that is, isolated handling of message exchanges, and administration and management capabilities offered on a per tenant basis. For the purposes of this work we focused on the latter aspect. More specifically, we first provided a short introduction to JBI and its relationship to the ESB technologies. Then we proceeded to identify a set of functional and non-functional requirements for multi-tenancy awareness for ESBs based on our experiences and literature. Based on this discussion, we presented a generic multi-tenant aware administration and management framework based on the JBI specification that satisfies these requirements. As a proof-of-concept realization of our proposal we instantiated the architectural framework in the JBIMulti2 solution. JBIMulti2 is based on Apache ServiceMix and demonstrates the efficacy of our proposal by allowing for the successful implementation of multi-tenant aware administration and management capabilities. By means of an ESB benchmark, we demonstrated that the performance of our solution is comparable to that of a non multi-tenant aware implementation of ServiceMix for a large number of messages and concurrent users.

Currently, we are in the process of providing a more thorough performance evaluation along the lines we discussed in Section IV. In particular, we are interested in demonstrating the direct benefits of multi-tenancy on the level of ESBs by comparing performance against the cost of using multi-tenant and non multi-tenant aware ESBs. Toward this goal, we are also working on improving and optimizing the performance of our implementation. Furthermore, we also aim to address the various shortcomings that we have identified in the previous sections (i.e. finalizing the Web GUI component, enabling horizontal scalability through a load balancer, and realizing multi-tenant aware dynamic service selection and discovery).

We also plan to take advantage of using the JBI specification as the basis of our architectural framework and apply the same techniques and architectural solutions to non-ESB solutions, like for example application servers, that comply with this specification. Finally, in this work we scoped the discussion to ensuring the isolation of data between different customers of the ESB. Ensuring performance isolation (as defined by NFR_1) however is an equally important requirement for multi-tenancy with many implications about the various customers of the ESB, see for example [7]. For this purpose, we plan to investigate to what extent our proposed framework is able to ensure performance isolation, and what mechanisms are required to be put in place.

Acknowledgments

The research leading to these results has received funding from the 4CaaSt project (http://www.4caast.eu) part of the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258862.

The authors would like to thank the people at Flexiant, and in particular Craig Sheridan and Tabs Sharif for their invaluable help with the evaluation environment.

References

[1] D. A. Chappell, *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[2] N. Josuttis, *SOA in Practice*. O'Reilly Media, Inc., 2007.

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[4] M. Behrendt *et al.*, "Introduction and Architecture Overview IBM Cloud Computing Reference Architecture 2.0," 2011.

[5] C. Guo *et al.*, "A Framework for Native Multi-Tenancy Application Development and Management," in *Proceedings of CEC/EEE'07*. IEEE, 2007.

[6] R. Mietzner *et al.*, "Combining Different Multi-Tenancy Patterns in Service-Oriented Applications," in *Proceedings of EDOC'09*. IEEE, 2009.

[7] R. Krebs *et al.*, "Architectural Concerns in Multi-Tenant SaaS Applications," in *Proceedings of CLOSER'12*, 2012.

[8] S. Walraven *et al.*, "A Middleware Layer for Flexible and Cost-Efficient Multi-tenant Applications," *Middleware'11*, 2011.

[9] Java Community Process, "Java Business Integration (JBI) 1.0, Final Release," 2005.

[10] R. Chinnici *et al.*, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," 2007.

[11] A. Azeez *et al.*, "Multi-tenant SOA Middleware for Cloud Computing," in *Proceedings of CLOUD'10*, 2010.

[12] F. Chong, G. Carraro, and R. Wolter, "Multi-Tenant Data Architecture," MSDN, 2006.

[13] D. Pritchett, "BASE: An ACID Alternative," *Queue*, vol. 6, no. 3, 2008.

[14] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*. Addison Wesley, 2005.

[15] G. Hohpe and B. Woolf, *Enterprise Integration Patterns*. Addison-Wesley Professional, 2003.

[16] R. S. Sandhu *et al.*, "Role-based Access Control Models," *Computer*, vol. 29, 1996.

[17] OSGi Alliance, "OSGi Service Platform: Core Specification Version 4.3," 2011.

[18] Java Community Process, "Enterprise JavaBeans (EJB) 3.0, Final Release," JSR-220, 2006.

[19] 4CaaSt Consortium, "Immigrant PaaS Technologies: Scientific and Technical Report D7.1.1," Deliverable, 2011.