Institute of Architecture of Application Systems

ESB^{MT}: Enabling Multi-Tenancy in Enterprise Service Buses

Steve Strauch*, Vasilios Andrikopoulos*, 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, Frank Leymann, and
               Dominik Muhler},
  title     = {ESB^MT: Enabling Multi-Tenancy in Enterprise Service Buses},
  booktitle = {Proceedings of the 4th IEEE International Conference on Cloud
               Computing Technology and Science, CloudCom 2012,
               3-6 December 2012, Taipei, Taiwan},
  year      = {2012},
  pages     = {456-463},
  publisher = {IEEE Computer Society}
}
```


Universität Stuttgart
Germany

2012 IEEE 4th International Conference on Cloud Computing Technology and Science

ESB^{MT}: Enabling Multi-Tenancy in Enterprise Service Buses

Steve Strauch, Vasilios Andrikopoulos, 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*—Multi-tenancy is an essential property of Cloud computing. It helps service providers to maximize resource utilization and reduce servicing costs per customer. It is therefore important for key components of the contemporary enterprise environment like the Enterprise Service Bus (ESB) to support and enable multi-tenancy. For this purpose, in this work we investigate the requirements for multi-tenant ESB solutions, propose an implementation-agnostic ESB architecture that addresses these requirements, and discuss our proof-of-concept realization of this architecture.**

***Keywords*-Multi-tenancy; Enterprise Service Bus; Cloud-enabled middleware**

I. Introduction

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 focus of this work.

Two different consumer types have to be distinguished 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.

Multi-tenancy has been defined in different ways in the literature, see for example [1], [2], [3]. Such definitions however do not address the whole technological stack behind the different Cloud service models [4] (IaaS — Infrastructure as a Service, PaaS — Platform as a Service, SaaS — Software as a Service). For this purpose, in this paper 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.* This definition affects the different Cloud service models in different ways. In this work we focus solely on the PaaS model and investigate how multi-tenancy can be enabled for platforms offered as a service. In particular, we show how a critical middleware component of the PaaS model like the Enterprise Service Bus (ESB) [5] can be made multi-tenant, irrespective of the particular implementation technology used (i.e. which ESB solution is used). The concept of ESB as the messaging hub between applications addresses the fundamental need for application integration and in the last years it has become ubiquitous in enterprise computing environments. ESBs control the message handling during service invocations and are at the core of each Service-Oriented Architecture (SOA) [5], [6]. In order therefore to leverage the transition of enterprise environments to the Cloud paradigm, it is essential to make ESBs multi-tenant.

The contributions of this paper can be summarized as follows:

1) An identification of the requirements of enabling multi-tenancy for ESB solutions.
2) A proposal for a implementation-agnostic ESB architecture called ESB^{MT} that fulfills these requirements.
3) A prototype implementation of ESB^{MT}.

The rest of this paper is organized as follows: Section II motivates this work by means of an informative scenario. Drawing from this scenario, Section III proceeds to identify and discuss the requirements for multi-tenancy in ESB solutions as a part of the PaaS model. These requirements are addressed in Section IV, which presents ESB^{MT}, a generic, implementation independent ESB architecture. Section V discusses the realization of this architecture as a proof-of-concept implementation of our proposal and the evaluation of this realization; Section VI compares it with existing works. Finally, Section VII summarizes our findings and briefly presents future work.

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

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

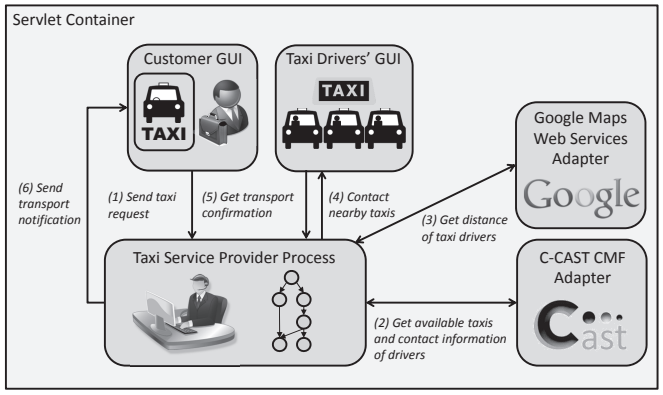

Figure 1. Communication of Web applications of Taxi Scenario without ESB

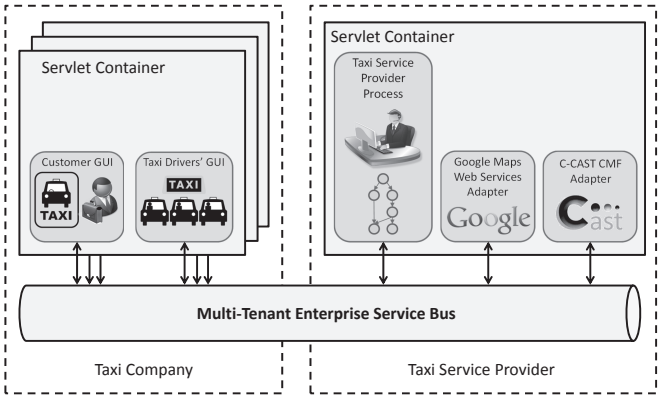

Figure 2. Communication of Web applications of Taxi Scenario after introducing multi-tenant ESB

II. Motivating Scenario

The European Union research project 4CaaSt[2] aims to create a Cloud platform to design services and compositions based on Cloud-aware building blocks provided by the platform, offer them in a marketplace, and operate them at Internet-scale. The goal of the 4CaaSt platform is to lower the entry barrier for small and medium enterprises by offering an advanced environment, which reduces the effort to create innovative applications leveraging the benefits of Cloud computing. In the scope of 4CaaSt, the *Taxi Scenario* use case has been defined, where a service provider offers a taxi management software as a service to different taxi companies, i.e., *tenants*. Taxi company customers, who are the *users* of the tenant, submit their taxi transportation requests to the company that they are registered with. The taxi company uses the taxi management software to contact nearby taxi drivers. Once one of the contacted taxi drivers has confirmed the transportation request, the taxi management software sends a transport notification containing the estimated arrival time to the customer.

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

Figure 1 provides an overview of the realization of the taxi scenario without using an ESB. The taxi management software is implemented as a BPEL process [7]. The BPEL process leverages the Context Casting Context-Management Framework (C-CAST CMF)[3], which provides context information about taxi cab locations and taxi driver contact details. Moreover, Google Maps Web Services [8] provide distance calculations between the location of a taxi cab and the pick up location. All components of the taxi booking service are Web applications deployed in a Servlet Container. Additionally, the BPEL engine Orchestra [9] is deployed as a Web application, executing the taxi service provider process. As the service endpoints of the C-CAST CMF and the Google Maps Web Services are incompatible with the BPEL process, two adapter applications mediate between the BPEL process and these external services. All applications communicate via point-to-point messaging connections.

Introducing an ESB as the messaging middleware (Figure 2) enables loose coupling and provides a more flexible integration solution by avoiding hard-coded point-to-point connections. This makes the monitoring, management, and maintenance of the taxi application easier and more effective. Furthermore, enabling multi-tenancy at the ESB level allows *multiple* taxi companies to use the same taxi application offered as a service by a *single* provider (using customized GUIs for their customers and drivers if required) as shown in Figure 2. Apart therefore from allowing taxi companies to outsource the development, deployment, operation, and management of such an application to a service provider, this solution also maximizes the benefits on the provider side. For this reason, the 4CaaSt project takes special care in developing a multi-tenant ESB as an essential building block of its PaaS offering.

III. Requirements for Multi-tenant ESBs

In the following, we discuss the requirements for multi-tenancy of ESBs solutions as a key component of the PaaS model. For this purpose we first discuss how multi-tenancy affects the PaaS model in general, before refining the discussion further for ESBs.

A. Multi-tenancy in the PaaS Delivery Model

Discussing multi-tenancy requires that the views of all involved parties are considered, namely both the providers and the consumers of multi-tenant aware services and applications. From the providers' point of view, multi-tenancy allows to maximize the utilization of provided resources and therefore enables maximization of profit. For service consumers, multi-tenancy has to be largely transparent, apart from providing access credentials when using the service or application. More importantly, consumers must have the impression that they are the only ones using the multi-tenant

[3]The C-CAST project: http://www.ict-ccast.eu

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

service or application, without suffering from side effects caused by other consumers regarding, e.g., quality of services. Finally, consumers need to be provided with customization capabilities, such as taxi company-specific Web interfaces in the Taxi Scenario.

The three Cloud service models (I-,P- and SaaS) differ significantly in the granularity of the functionality provided to the consumer, and the required capability of the consumer to manage and control the underlying Cloud infrastructure. The responsibility of the provider and the effort of the consumer to enable multi-tenancy is therefore different, depending on the chosen Cloud service model. PaaS in particular, is the Cloud service model where the responsibility and effort of provider and consumer are nearly the same with respect to our definition of multi-tenancy. The exact effort of the consumer depends on the scenario and the application to be realized. The consumer is responsible to enable multi-tenancy of the application and the corresponding artifacts deployed on the platform; for example, the database schema used has to support multi-tenancy natively [10]. The provider has to enable multi-tenancy for the hardware resources and infrastructure, as well as the platform on which the various applications are deployed. Furthermore, for the sake of backward compatibility the deployment of non multi-tenant applications has also to be possible, otherwise the target community of the PaaS offering will be limited. Therefore, the service offered via PaaS by the provider has to support the deployment of both multi-tenant and non multi-tenant services and applications.

B. ESB Multi-tenancy 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 ESBs. Toward this goal we also refined the multi-tenancy characteristics (e.g. tenant awareness) identified in the literature, e.g. [11], [1], [2], [3].

Functional requirements: The following functionalities must be offered by any multi-tenant ESB.

- FR_1 *Tenant awareness*: An ESB 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 deployment and configuration*: The deployment and configuration of the ESB and the services available for a certain tenant should be managed in a transparent manner by the ESB.
- 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 ESB solution 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 services.
- FR_5 *Backward compatibility*: The ESB solution should be able to used seamlessly and transparently by services and applications that are not multi-tenant aware.

Non-functional requirements: In addition to the required functionalities, multi-tenant ESBs 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. ESB 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).

Both functional and non-functional requirements are taken into consideration for the design of the architecture we present in the following section.

IV. A Multi-tenant Aware ESB Architecture

Figure 3 provides an overview of our proposal for a generic multi-tenant ESB architecture (ESB^{MT}), which fulfills the requirements identified in the previous section. More specifically, the three layer ESB^{MT} 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 in a bottom-up fashion.

A. Resources layer

The Resources layer consists of an *ESB Instance Cluster* and a set of registries. The ESB Instance Cluster bundles together multiple *ESB Instances*. Each one of these Instances performs the tasks usually associated with traditional ESB solutions, that is, message routing and transformation. In the simplest case, the ESB Instance Cluster may consist of only one (running) ESB Instance handling all tenants and users using an ESB^{MT} implementation. Since this however may create performance issues, in the ESB^{MT} architecture a clustering mechanism similar to the one provided for example by Apache ServiceMix [12] is recommended.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

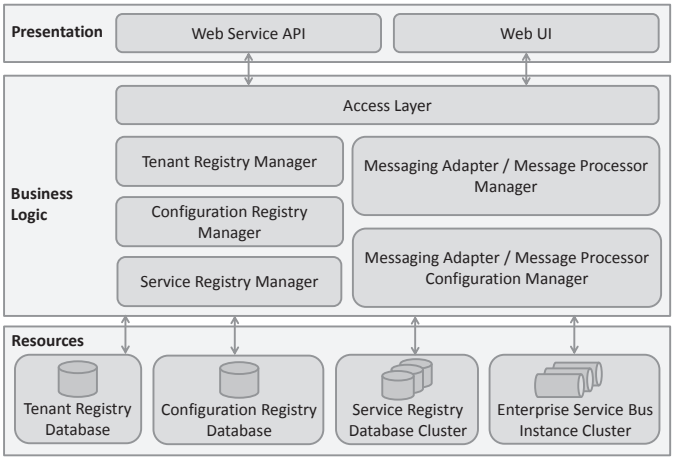

Figure 3. Overall ESB^{MT} Architecture

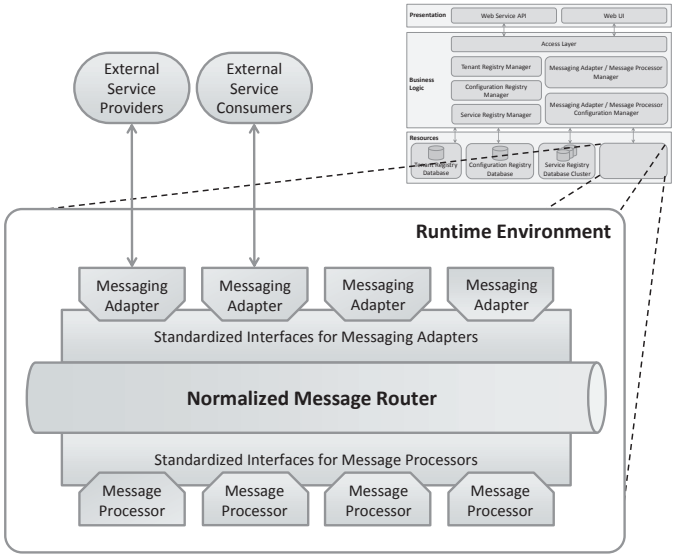

Figure 4. Architecture of an ESB Instance

Each ESB Instance consists of three main components: a *Normalized Message Router*, *Messaging Adapters*, and *Message Processors* (Figure 4, zooming in on the bottom right part of Figure 3). Messaging Adapters are responsible for handling the communication with external services and applications (*External Service Providers* and *Consumers* in Figure 4) and converting to and from a normalized internal format for all incoming and outgoing messages, respectively. Message Processors provide additional business logic internal to the ESB related to message processing such as routing. The Normalized Message Router takes care of the internal routing between Messaging Adapters and/or Message Processors. These components appear under different names in many existing ESB solutions.

In order to enable multi-tenancy in any ESB solution, these components, and all other components in the ESB architecture, must be *multi-tenant aware*, i.e. able to operate with multiple tenants and users using the same instance of the ESB. For an ESB Instance in particular, this means that Adapters and Processors are able to handle messages containing tenant and user information, and process such messages accordingly in a multi-tenant manner (FR_1). For example, a message may be routed to different endpoints based on the tenant information contained in the message. Additionally, message flows of tenants and users when communicating with an ESB Instance, as well as message flows inside the ESB Instance, must be isolated from the message flows of other tenants and users (NFR_1). Furthermore, the Messaging Adapters and Message Processors have to be able to support tenant- and user-specific configurations when required (FR_2). This enables for example, that for each tenant a new endpoint for communication with the protocol specific adapter is created during configuration (NFR_1). The deployment/undeployment and configuration of Messaging Adapters and Message Processors in an ESB Instance is performed by means of a set of *standardized interfaces*. While these interfaces, and all other components of the ESB 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 Adapters and Processors must still be possible. Processing and routing of non multi-tenant aware messages has to be performed normally, by assigning them for example into a default tenant for this purpose.

Going back to Figure 3, within the Resource layer we also introduce three different types of registries. The *Service Registry* stores the services registered with the various ESB Instances, as well as the configuration of the Messaging Adapters and Message Processors installed in each ESB Instance in the ESB Instance Cluster (Figure 4) in a tenant-isolated manner [10] (FR_2). Currently we are focusing on the approach that each ESB instance of the ESB Instance Cluster has the same messaging adapters and message processors installed. As the messaging adapters and message processors are common, and in order to offer the possibility of horizontal scalability support [13], a load balancer (not shown in Figure 3) must retrieve the required configurations from the Service Registry and deploy them when starting an additional ESB 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). The *Configuration Registry* stores all configuration data

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

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 ESB Instances (FR_2), the mapping of ESB 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). When a tenant or user interacts with the multi-tenant ESB system, 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_1).

B. 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 [1] based on role-based access control [15]. The tenants and their corresponding users have to be identified and authenticated once when the interaction with the ESB 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 ESB 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* (Figure 3) encapsulating the functionality to manage and interact with the 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 in the Resources layer. The *Messaging Adapter/Message Processor Managers* deploy and undeploy Messaging Adapters and Message Processors in each ESB Instance in the Cluster, while the *Configuration Managers* take care of configuring them appropriately. Both managers are using the standardized interfaces provided by each ESB Instance for this purpose (Figure 4), as discussed in the Resources layer.

C. Presentation layer

The Presentation layer contains two components allowing the customization, administration, management, and interaction with an ESB$^{\mathrm{MT}}$ implementation: 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 Web UI. A discussion on the particular mechanisms to be used for this purpose is outside of the scope of this work.

V. Realization & Evaluation

A proof-of-concept realization of the ESB$^{\mathrm{MT}}$ architecture is provided as a deployment diagram in Figure 5. The realization is based on the open source ESB Apache ServiceMix version 4.3.0 (hereafter referred to simply as ServiceMix) [12] and components and libraries being reused are marked in gray. ServiceMix is based on the OSGi Framework [16]. OSGi bundles realize the ESB functionality complying to the JBI specification [17].

ServiceMix is provided with several JBI components. Binding Components (BCs) are Messaging Adapters (in the sense of Figure 4) supporting various protocols such as SOAP over HTTP, FTP, or JMS. Service Engines (SEs) are JBI components providing additional business logic within the ESB. For example, the SE for Apache Camel [18] enables usage of Enterprise Integration Patterns [19]. In this sense they serve as the Message Processors in our architecture.

The original ServiceMix BC for HTTP version 2011.01 and the original Apache Camel SE version 2011.01 were extended in our prototype in order to support multi-tenancy (see the *servicemix-http-mt* and *servicemix-caml-mt* components in Figure 5). 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 (Figure 5). 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 ESB$^{\mathrm{MT}}$ (Figure 3) and is running in the Java EE 5 application server JOnAS version 5.2.2 [20], 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 [21].

As many JBI containers deployed on several ServiceMix instances can be involved in the distributed transactions, and the distributed transaction can contain many JBI containers, this might lead to a performance bottleneck. Hence, the Web application subdivides the transaction to the JBI containers using messaging with guaranteed delivery [19]. If the message is persistently stored in the message topic, the distributed transaction will commit. Afterwards, each JBI container

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

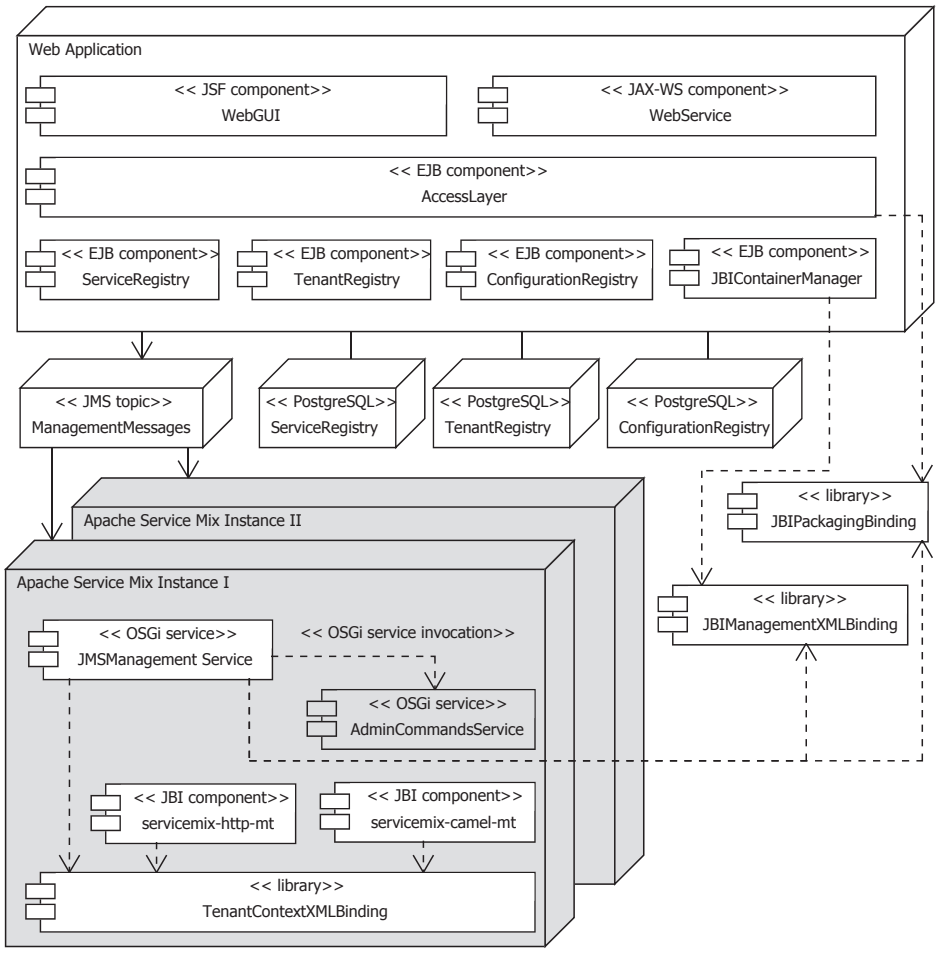

Figure 5. Deployment diagram of prototype realization of ESB^{MT}

acts as selective, transactional, and durable subscriber. A transaction between each corresponding JBI container and the topic ensures that the message is successfully processed before being deleted from the topic. For JMS messaging we use Apache ActiveMQ version 5.3.1 [22]. The *ServiceRegistry*, *TenantRegistry*, and *ConfigurationRegistry* components are realized based on PostgreSQL version 9.1.1 [23]. The *AccessLayer* of the Web Application applies the Session Façade pattern [24], which is a design pattern for EJB projects encapsulating business logic in order to minimize the number of calls to the EJB container. The Web Service API of the Web Application is based on the Java API for XML-Based Web Services version 2.0 [25]. The *WebGUI* has been specified and designed based on JavaServer Faces version 1.2 [26], but the implementation is still ongoing.

The evaluation of the realization of the architecture within the context of 4CaaSt is based on the Taxi Scenario introduced in Section II. For this purpose, we implemented the motivating scenario discussed in Section II for two taxi companies (tenants). Both companies are using the same taxi management application hosted by the 4CaaSt platform. The application provides an interface for their registered customers and drivers (users) that is customizable by the companies on demand. Using this interface, the customer can request a taxi by providing the necessary information through, e. g., a smartphone device.

The customer request is then forwarded to the two nearest drivers and pops up in their GUIs (as shown in Figure 6). The first driver that confirms the request is assigned to the customer. The driver further has the option to get routing information to the designated pick up location through an integration with Google Maps Web Services. Based on the distance between the driver and the pick up location, the customer receives a notification containing the estimated pick up time. All messaging between services in the scenario as shown in Figure 2 is handled by our realization of the ESB^{MT} architecture discussed above. A video demonstrating the taxi application in action is available at http://tiny.cc/4caast_taxi_video. We are currently in the process of evaluating the performance of our ESB^{MT}

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

Figure 6. Screenshot of the GUI used by the taxi drivers of one company

realization (see Section VII).

VI. Related Work

The central component of each SOA realization is the ESB, which connects service providers to service consumers, mediates messages between them, and supports the essential concept of loose coupling [5], [6]. For this reason, in [27] we surveyed a number of existing ESB solutions and evaluated their multi-tenancy. More specifically, we evaluated Apache ServiceMix, Microsoft BizTalk Server, JBoss ESB, Mule ESB, OW2 Petals ESB, IBM WebSphere ESB, and WSO2 ESB.

Our investigation showed that the surveyed ESB solutions in general lack in support of multi-tenancy [27]. Even in the case of products like IBM WebSphere ESB [28] and WSO2 ESB [29] 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 Axis2 [30] 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.

In addition to ESB solutions we also investigated the PaaS offering Force.com operated by Salesforce.com [31] regarding multi-tenancy. Force.com is an application development platform focusing on multi-tenancy requirements tenant-based deployment and configuration as well as performance isolation. In contrast to the architecture we propose, Force.com is a meta data-driven architecture. Thus, when a customer requests for example the business application of a tenant, all components of the application are created dynamically based on meta data. Sharing is therefore achieved only on platform level, and not on application level, which makes this solution not truly multi-tenant based on our definition of multi-tenancy.

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

VII. Outlook and Future Work

Multi-tenancy on the PaaS level allows service providers to offer multiple tenants customizable versions of the same version for consumption by their users. ESB solutions have become ubiquitous in the last years for enterprise environments and as such enabling multi-tenancy for ESBs is essential. The requirements for this effort, as identified by the literature and in the context of European Union's project 4CaaSt, span from functional (e.g. tenant awareness and tenant-specific interfaces) to non-functional (e.g. tenant isolation). In order to address these requirements we propose the generic, implementation-agnostic ESB^{MT} architecture in three layers (presentation, business logic, and resources), which also allows for extensibility and reusability of its components by other PaaS building blocks, e.g. orchestration engines. As we demonstrate by means of a proof-of-concept realization, ESB^{MT} can be realized in a straightforward manner using open source technologies, validating our approach and allowing for further evaluation of our work in the future.

In particular, we are already working on evaluating the performance of our multi-tenant ESB implementation using different workload profiles for the Taxi Scenario application, and compare it with that of a similar non multi-tenant ESB solution under equivalent load. In addition, for the same workloads we will also monitor and compare the resource requirements for multi-tenant and non multi-tenant solutions so we can demonstrate quantitatively the benefits of multi-tenancy on the ESB level. Furthermore, and for purposes of completeness, we are working on finalizing the implementation of the Web GUI component as discussed in Section V. Finally, we plan to investigate how horizontal scalability [13] can also be enabled for solutions implementing the ESB^{MT} architecture in order to offer a fully Cloud-ready ESB solution.

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 company, product and service names used in this publication are for identification purposes only. All trademarks

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

and registered trademarks are the property of their respective owners.

REFERENCES

[1] C. Guo, W. Sun, Y. Huang, Z. Wang, and B. Gao, "A Framework for Native Multi-Tenancy Application Development and Management," in *Proceedings of the 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'07)*. IEEE, 2007.

[2] R. Mietzner, T. Unger, R. Titze, and F. Leymann, "Combining Different Multi-Tenancy Patterns in Service-Oriented Applications," in *Proceedings of the 13th IEEE Enterprise Distributed Object Conference (EDOC 2009)*. IEEE, September 2009.

[3] R. Krebs, C. Momm, and S. Konev, "Architectural Concerns in Multi-Tenant SaaS Applications," in *Proceedings of the 2nd International Conference on Cloud Computing and Service Science (CLOSER'12)*. SciTePress, April 2012.

[4] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," September 2011. [Online]. Available: http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909616

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

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

[7] A. Alves *et al.*, "Web Services Business Process Execution Language Version 2.0," Comitee Specification, April 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[8] Google, Inc., "Google Maps API Web Services." [Online]. Available: http://code.google.com/intl/en/apis/maps/documentation/webservices/

[9] OW2 Consortium, "Orchestra: Open Source BPEL / BPM Solution." [Online]. Available: http://orchestra.ow2.org

[10] F. Chong, G. Carraro, and R. Wolter, "Multi-tenant data architecture," MSDN, 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479086.aspx

[11] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, "Multi-tenant SOA Middleware for Cloud Computing," in *Proceedings of IEEE 3rd International Conference on Cloud Computing (CLOUD'10)*, July 2010.

[12] Apache Software Foundation, "Apache ServiceMix." [Online]. Available: http://servicemix.apache.org

[13] D. Pritchett, "BASE: An ACID Alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.

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

[15] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based Access Control Models," *Computer*, vol. 29, pp. 38–47, February 1996.

[16] OSGi Alliance, "OSGi Service Platform: Core Specification Version 4.3," 2011. [Online]. Available: http://www.osgi.org/Download/Release4V43/

[17] Java Community Process, "Java Business Integration (JBI) 1.0, Final Release," 2005, jSR-208. [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr208/

[18] Apache Software Foundation, *Apache Camel User Guide 2.7.0*, 2011. [Online]. Available: http://camel.apache.org/manual/camel-manual-2.7.0.pdf

[19] Gregor Hohpe and Bobby Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.

[20] OW2 Consortium, "JOnAS: Java Open Application Server." [Online]. Available: http://wiki.jonas.ow2.org

[21] Java Community Process, "Enterprise JavaBeans (EJB) 3.0, Final Release," JSR-220, 2006. [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr220/

[22] Apache Software Foundation, "Apache ActiveMQ." [Online]. Available: http://activemq.apache.org

[23] PostgreSQL Gobal Development Group, "Postgresql." [Online]. Available: http://www.postgresql.org

[24] F. Marinescu, *EJB Design Patterns: Advanced Patterns, Processes, and Idioms*. John Wiley & Sons, Inc., 2002.

[25] Java Community Process, "The Java API for XML-Based Web Services (JAX-WS) 2.0, Final Release," JSR-224, 2006. [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr224/

[26] ——, "JavaServer Faces Specification (JSF) 1.2, Final Release," JSR-252, 2006. [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr252/

[27] 4CaaSt Consortium, "Immigrant PaaS Technologies: Scientific and Technical Report D7.1.1," Deliverable, July 2011. [Online]. Available: http://www.4caast.eu/wp-content/uploads/2011/09/4CaaSt_D7.1.1_Scientific_and_Technical_Report.pdf

[28] IBM, "IBM WebSphere ESB." [Online]. Available: http://ibm.com/developerworks/webservices/library/ws-multitenant/

[29] WSO2, "WSO2 Enterprise Service Bus." [Online]. Available: http://wso2.com/products/enterprise-service-bus/

[30] Apache Software Foundation, "Apache Axis2." [Online]. Available: https://axis.apache.org/axis2/java/core/

[31] C. D. Weissman and S. Bobrowski, "The Design of the Force.com Multitenant Internet Application Development Platform," in *Proceedings of SIGMOD International Conference on Management of Data (SIGMOD'09)*. ACM, 2009.

All links were last followed on October 5, 2012.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE