# SCE$^{MT}$: A Multi-tenant Service Composition Engine

Michael Hahn, Santiago Gómez Sáez, Vasilios Andrikopoulos,
Dimka Karastoyanova, Frank Leymann

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

BibTeX:

```
@inproceedings{INPROC-2014-69,
  author    = {Michael Hahn and Santiago G{\'o}mez S{\'a}ez and Vasilios
               Andrikopoulos and Dimka Karastoyanova and Frank Leymann},
  title     = {{SCE^MT: A Multi-tenant Service Composition Engine}},
  booktitle = {Proceedings of the 7th International Conference on Service-
               Oriented Computing and Applications, SOCA 2014, 17-19 November
               2014, Matsue, Japan},
  year      = {2014},
  pages     = {89--96},
  publisher = {IEEE Computer Society}
}
```

# SCE$^{\text{MT}}$: A Multi-tenant Service Composition Engine

Michael Hahn, Santiago Gómez Sáez, Vasilios Andrikopoulos, Dimka Karastoyanova and Frank Leymann
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
Email: {firstname.lastname}@iaas.uni-stuttgart.de

*Abstract*—**The support of multi-tenancy is an essential requirement for leveraging the full capacity of Cloud computing. Multi-tenancy enables service providers to maximize the utilization of their infrastructure and to reduce the servicing costs per customer, thus indirectly benefiting also the customers. In addition, it allows both providers and consumers to reap the advantages of Cloud-based applications configurable for the needs of different tenants. Nowadays, new applications or services are typically compositions of multiple existing services. *Service Composition Engines* (SCEs) provide the required functionality to enable the definition and execution of such compositions. Multi-tenancy on the level of SCEs allows for both process model, as well as underlying infrastructure sharing. Towards the goal of enabling multi-tenancy of SCEs, in this paper, we investigate the requirements and define a general architecture for the realization of a multi-tenant SCE solution. This architecture is prototypically realized based on an open-source SCE implementation and integrated into an existing multi-tenant aware *Enterprise Service Bus* (ESB). The performance evaluation of our prototype shows promising results in terms of the degradation introduced due to processing and communication overhead.**

## I. Introduction

Cloud solutions provide IT resources in an agile manner over the Internet in a pay-per-use model resulting into numerous advantages when outsourcing applications and services into the Cloud, e.g. reduced operational costs and rapid elasticity for enterprises. To benefit from the full potential of Cloud computing, outsourced applications and services need to be *multi-tenant aware* [1], [2]. Multi-tenancy implies that applications should be designed so that they maximize the resource sharing between multiple consumers. Thus service providers are able to maximize the resource utilization and as a result reduce their servicing costs per customer [3]. Multi-tenancy is a main prerequisite to enable other very important characteristics of applications or services like *isolation*, *configurability* and *scalability* [1], [4], [5]. There is a variety of different multi-tenancy definitions in literature, for example in [2], [4] or [5]. For this paper we use the definition provided in [3] as a basis, where multi-tenancy is defined as *the sharing of the whole technological stack (hardware, operating system, middleware and application instances) at the same time by different tenants (organizational units) and their corresponding (tenant) users*.

In this work we focus on the multi-tenancy of middleware, and in particular on the sharing of a single *Service Composition Engine* (SCE) instance among multiple consumers, positioned therefore in the Platform as a Service (PaaS) Cloud delivery model. This sharing manifests both in terms of multiple tenants using the same engine in an isolated manner, but also as different combinations of tenants and/or tenant users sharing the same process model, potentially with different configuration options per tenant as discussed e.g. in [6]. Previous work [7] demonstrated that the concept of SCE as a Service is both feasible and efficient, however by delegating the problem of multi-tenant awareness on the level of the messaging middleware used. In this work, we instead propose a generic, reusable architecture for multi-tenant aware SCEs that separates the composition aspect from tenant management. As discussed in the following sections, this approach returns results similar to the ones reported by [7], while allowing for multiple potential realizations and multi-tenancy on the level of both process model and engine instance.

More specifically, the contributions of this paper can be summarized as follows:

1) An identification of the requirements for a multi-tenant aware SCE solution.
2) A generic, implementation-agnostic SCE architecture of a multi-tenant Service Composition Engine, addressing these requirements.
3) A prototypical realization (SCE$^{\text{MT}}$) and performance evaluation of the proposed architecture.

The rest of this paper is structured as follows: Section II discusses the desired behavior of multi-tenant SCE solution, based on which, and in combination with a literature survey, a set of requirements are extracted. These requirements are used to drive the design of a multi-tenant aware SCE architecture (Section III), a prototype implementation of which (SCE$^{\text{MT}}$) is presented in Section IV. Section V focuses on evaluating the performance of SCE$^{\text{MT}}$. Finally, the paper concludes with related work (Section VI), and a summary of our findings together with a short outlook on future work (Section VII).

## II. Requirements on a Multi-tenant SCE

The main requirement in a multi-tenant environment is to be able to correctly identify the resources associated with a tenant, both in terms of exchanged and persisted data, as well as underlying computational infrastructure. This is the basis for all higher-level functionalities like configurability, isolation and accounting. To uniquely identify a tenant or tenant user in the rest of the discussion we use the concept of *tenant context* [3], [8], as e.g. a combination of a unique tenant and user ID. To distinguish if an operation (e.g. deployment, administration, invocation) is executed by a tenant or tenant user, we then use the term “under tenant context”.

### A. Behavior of a Multi-tenant SCE

The following general for all SCEs use cases have to be considered with respect to the required behavior of a multi-tenant SCE:

*Access Control Layer:* A multi-tenant SCE requires an access control layer (ACL) which (1) is able to identify a tenant (or tenant user) and (2) provides an access control mechanism that checks if an identified user has the required access rights to use a specific part of the functionality of an SCE like deployment, management or instantiation of a process model. For this, the mechanism compares the tenant information associated to the resource to be processed (e.g. a process model on process instantiation) and the identified tenant using e.g. the tenant context.

*Process Deployment:* After process models are defined they have to be deployed on an SCE to enable their execution. For this purpose, the so called *Deployment Bundles* are generated, containing a set of process models and related files like service interface descriptions (e.g. using WSDL[1]), required data models (e.g. specified as XSD[2]) or some required SCE-specific information (e.g. endpoints of services) collected in a deployment descriptor. If such a Deployment Bundle is deployed by a tenant user to a multi-tenant aware SCE, all contained resources/artifacts must be associated to the tenant user. This is realized inside the SCE by attaching tenant information to all contained resources of a Deployment Bundle. The related data are persisted in a multi-tenant aware *Model Database*. Independent of the underlying implementation (e.g. database, file system), this Model Database must be able to isolate the Deployment Bundles of different tenant users based on the assigned tenant information. Chong et al. provide three main strategies for data isolation on the database level [9]: separate databases, shared database/separate schemas, and shared database/shared schemas.

*Process Instantiation:* Upon process instantiation, the SCE should be able to identify which message belongs to which tenant user in order to uniquely identify the initiator of a process instance. This enables the ACL to check if an incoming request sent to a process service should be processed by a new process instance, or whether the request should be immediately rejected. Therefore, the ACL compares the tenant user associated to the incoming request with the tenant user associated to the process model stored in the Model Database. In other words, the engine has to check if the sender of the request is allowed to instantiate the process model. If the tenant information refers to the same tenant user, the ACL can forward the request to the service interface of the process model. If the tenant information is not the same, the ACL returns a corresponding fault message to the requester. The generated new process instance is directly associated with the tenant user to whom the request belongs. This is necessary to allow the ACL later to authenticate calls to the Management Interfaces for an instance, like suspending or terminating the instance. If a tenant has registered any configuration data for the instantiated process model, the data are dynamically loaded from a multi-tenant aware *Configuration Database* and assigned to the instance. The runtime data of all process instances are persisted in a multi-tenant aware *Runtime Database*.

[1]Web Services Description Language (WSDL) 1.1: http://www.w3.org/TR/wsdl

[2]XML Schema Definition Language (XSD) 1.1: http://www.w3.org/TR/xmlschema11-1/

*Configurability:* Due to the fact that the SCE is shared between multiple tenant users, it should support configuration possibilities. This means it should allow tenant users to customize some parts of the engine to their needs without the necessity to change the underlying implementation of the SCE. In order to enable also the sharing of the enacted process models, in addition the SCE should provide configurability on the process model level. This enables the customization of process models on a tenant user basis by specifying configuration data while the underlying model remains unchanged. Therefore, the configuration data has to be stored in a multi-tenant aware *Configuration Database* (as in the Process Instantion case) to enforce its isolation on a tenant user basis.

*Service Invocation:* For the invocation of the composed services, the identification of the tenant user, on behalf of whom the message is sent, plays an important role. If the invoked service is also multi-tenant aware, the tenant information is used to check if the identified user has the permissions to invoke the service or not. For that purpose, the tenant information must be forwarded by the SCE to any invoked service whether it is multi-tenant or not. A non multi-tenant aware service will ignore the tenant information sent with the request message. A multi-tenant aware service will use the received tenant information to authenticate the incoming request, and may use it to support isolation and accounting of the utilized resources. Upon successful authentication, the service processes the request and replies a corresponding response message. If the authentication fails, the service rejects the incoming request and replies a corresponding fault message to the requester. Fault messages can then be handled by the SCE with the fault handling mechanisms of the underlying process execution language (e.g. languages like BPEL [10]).

*Process Instance Correlation:* A correlation context is usually used by SCEs to identify the correct instance of a process model to which the message should be forwarded, especially in the case of asynchronous communication. Enabling multi-tenancy does not affect the correlation mechanism of an SCE. The tenant information will be forwarded in all message exchanges and may be considered separately.

## B. Requirements

Based on the required SCE behavior, as discussed above, and the existing requirements found in literature, e.g. [2], [3], [4], [11], the following set of functional and non-functional requirements to be fulfilled by any multi-tenant aware SCE solution can be identified:

### 1) Functional Requirements:

- $FR_1$ *Tenant awareness:* An SCE must be able to identify multiple tenants and their associated resources like deployment bundles, process instances, messages, etc. This in turn requires enabling tenant-based authentication and role-based access control for tenants and their users.
- $FR_2$ *Tenant-based configuration and deployment:* The SCE should support tenant-based configurations of the engine itself and all deployed process models. Furthermore, the SCE must enable the deployment of process models on a per-tenant basis.

FR$_3$ *Tenant-specific interfaces:* A set of customizable Web Service interfaces and GUIs must be provided to enable the tenant-based management of SCE resources, like process models, instances or configuration data.

FR$_4$ *Shared registries:* Since the SCE solution might be integrated into an environment with other multi-tenant aware components demanding similar information, the SCE should work with a set of shared registries that contain data about tenants and their users, services and configurations.

FR$_5$ *Backward compatibility:* The SCE solution must be able to provide its functionality also to non multi-tenant aware applications.

*2) Non-functional Requirements:*

NFR$_1$ *Tenant Isolation:* Tenants and their resources must be isolated on all layers of the SCE to prevent the tenants from gaining access to resources of other tenants.

NFR$_2$ *Reusability & Extensibility:* The multi-tenancy enabling mechanisms and the underlying concepts must be agnostic to specific SCE implementations and technologies. The defined mechanisms, concepts and components should therefore be extensible, adaptable and reusable in different SCEs.

NFR$_3$ *Transparent integration:* Beyond the addition of the tenant context, the behavior and appearance of the SCE should be the same from a user's point of view, irrespective of how the multi-tenancy enabling mechanisms are realized.

NFR$_4$ *Scalability:* To enable horizontal scalability [12] (scale out) the SCE should run in a stateless fashion.

In the following section we present our proposal for a multi-tenant aware SCE architecture that addresses these identified requirements.

## III. Multi-tenant Aware SCE Architecture

In order to ensure the generality of the proposed approach, we not only abstract from the specifics of a particular SCE solution, but we also attempt to decouple as much as possible the operation of the SCE from the multi-tenancy management mechanisms that need to be put into place. For this purpose, we distinguish between two major components of the proposed architecture: an *SCE Multi-tenancy Manager* (SCE-MT Manager) and the *Multi-tenant Service Composition Engine* itself, as shown in Fig. 1. These two design decisions allow for the realization of our proposal by multiple SCE solutions, with the potential to be able to reuse an SCE-MT Manager across different SCE realizations.

### A. SCE-MT Manager

The SCE-MT Manager acts as a Multi-tenancy Enablement Layer (NFR$_2$, NFR$_4$) and provides the necessary support for administering and managing a clustered container potentially hosting different multi-tenant aware SCE implementations. It is a middleware component realizing the multi-tenancy functionality in a reusable way and therefore is not bound to a single SCE implementation. The concrete SCE implementations must only be adapted to integrate with the SCE-MT Manager. In addition, the SCE-MT Manager can be extended to provide

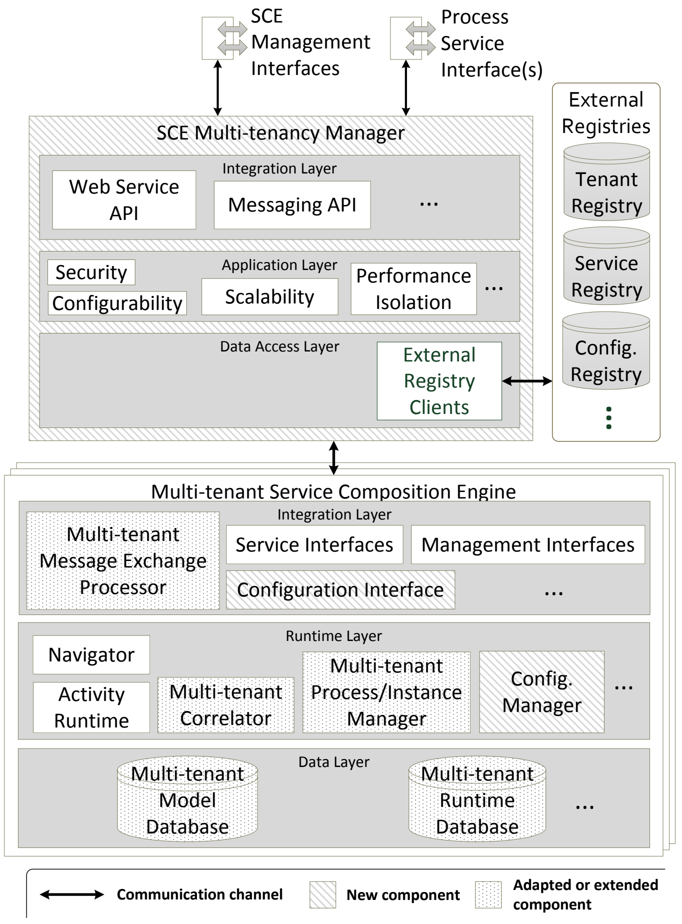


Figure 1. Architecture of a multi-tenant SCE with a separate multi-tenancy enablement layer

advanced functionality like performance isolation or scalability to all connected SCE implementations (NFR$_1$, NFR$_4$).

The SCE-MT Manager acts as an intermediate layer between a tenant user and an SCE. This means all requests sent to one of the SCE interfaces (*Management Interfaces* or *Process Service Interfaces* in Fig. 1) are consumed by the SCE-MT Manager which authenticates and reroutes them to the correct interface of one of the connected SCE instances (FR$_1$, NFR$_3$). For this purpose, the SCE-MT Manager may provide the SCE interfaces over a Middleware Container like an ESB to the outside. The authentication of all incoming requests sent to the SCE interfaces can then be handled over the ESB message routing mechanisms. On successful authentication the message is forwarded to the corresponding SCE service and in any other case the message is directly rejected.

The SCE-MT Manager follows a three layer architecture pattern [13]. The *Integration Layer* provides a *Web Service* and a *Messaging API* which enable the communication with the SCE-MT Manager, and the set of connected to it SCE instances (FR$_3$). An SCE can use the Messaging API or the Web Service to register itself with the SCE-MT Manager. The SCE-MT Manager also uses these APIs to communicate with all registered SCE instances, e.g. to send configuration data to an SCE if a new tenant is registered at the Tenant Registry. The *Data Access Layer* contains a list of clients

(*External Registry Clients*) which are used to integrate a set of distributed registries, like the *Service Registry*, the *Tenant Registry* or the *Configuration Registry* ($FR_4$). The distributed Service Registry stores process models independent of any SCE instance. The Tenant Registry provides the management of tenants and their users. Furthermore, the contained tenant data is used in all other registries to uniquely assign a resource (e.g. services, configuration data) to a tenant. The distributed Configuration Registry manages tenant-based configuration data, for example, configuration data for process models or an SCE instance registered by a tenant.

The *Application Layer* contains the following modules: The *Security* module contains all security related functionality and enables *Communication Isolation*, like the already described authentication of incoming messages ($NFR_1$). Furthermore, it realizes *Administration Isolation* by the authentication of any access to the distributed registries. The *Configurability* module handles the management of tenant-based configurations ($FR_2$). For example, on process instantiation it collects all necessary configuration data referenced by the calling tenant from the Configuration Registry and sends the data to the *Configuration Interface* of the corresponding SCE instance. The *Scalability* module provides functionality to enable scaling out like the dynamic creation of new SCE instances and can potentially provide a load balancing mechanism ($NFR_4$). The *Performance Isolation* module provides functionality to analyze the performance of all registered SCE instances (e.g. based on auditing data) and potentially reject new incoming requests of a tenant if he exceeds his quotas ($NFR_1$). More details on these two modules are out of the scope of this paper.

## B. Multi-tenant SCE

In order to allow the implementation of multi-tenancy in different SCE implementations, we abstract away from the specifics of SCE solutions like Apache ODE[3], WSO2 Business Process Server[4], OW2 Orchestra[5] or the YAWL Workflow Engine[6], and we compose logical components standing for specific SCE functionalities (e.g. navigation through the process model) in a unified architecture model. These logical components are also organized in three layers, as shown in Figure 1.

The *Integration Layer* exposes the engine-internal functionality to the outside and enables communication. It contains a *Message Exchange Processor* which handles the correct routing of incoming and outgoing messages to the their corresponding services (process models). For example, if a request is sent to the service interface of a process model which creates a new instance of the model, the Message Exchange Processor routes the request to the created instance and returns the response message of the instance back to the sender of the request. The *Service Interfaces* component provides the service endpoint for each deployed process model to the outside (e.g. as Web Service) and therefore enables sending requests to a process model. The *Management Interfaces* component provides the management functionality of the SCE to the outside (e.g. as a Web Service), like querying all deployed models or suspending a running instance.

The *Runtime Layer* contains the actual logic of the engine. The *Navigator* is responsible for the execution of process instances based on the control flow defined in the process model. The engine provides an Activity Runtime component for the functionality of each activity type of the underlying process model language. The *Correlator* component is responsible for the correlation of request and response messages. For example, if a process model has multiple concurrently running instances which invoke an external service, the response of this service has to be routed to the instance which sends the corresponding request. The *Process/Instance Manager* component contains the implementation of all process and instance management functionality which is provided over the Management Interfaces. This contains for example, the change of the execution state of an instance (e.g. suspend, resume, terminate), the deployment or undeployment of process models or querying some information from the engines databases, like a list of running instances.

The *Data Layer* provides the persistent stores of the engine which are typically realized using databases. The *Model Database* contains all process-related data, like a persisted version of all deployed process models, the state of each process model or a complete list of all started instances of each process. The *Runtime Database* contains all runtime-related data. This database is very important because it holds the instance contexts of all process instances, like variable data, incoming and outgoing messages or activity states. The Navigator and other components of the Runtime Layer use this instance contexts and the Runtime Database to provide their functionality. The Process/Instance Manager component also works with the databases. For example, undeploying a process model through the Process/Instance Manager causes the deletion of the corresponding process model from the Model Database.

Next we take a look at components that have to be adapted or are missing in conventional, non multi-tenant aware SCEs in order to enable multi-tenancy. The Message Exchange Processor must be extended to allow bi-directional tenant-aware communications ($FR_1$). Tenant-aware communications are supported by means of incorporating *tenant context* information in all incoming and outgoing messages, i.e. tenant and tenant user unique identifiers, and a key value structure for tenant specific additional information as shown in Figure 2. Such information enables the authentication and correlation of each incoming and outgoing message to their corresponding tenant and tenant user. The Message Exchange Processor should be able to handle seamlessly the communication with both non multi-tenant and multi-tenant aware services ($FR_5$). In addition to that, the Model and Runtime Databases should be extended to store any tenant-specific data in a tenant-aware manner [9] ($FR_1$). The Application Layer must also be adapted to support tenant-based configurations of the SCE ($FR_2$).

To enable the correlation of tenant-aware message exchanges and provide communication isolation ($NFR_1$), the Correlator has to be extended. The authentication of incoming messages sent to the Management Interfaces of an SCE can be handled in two ways. Either the messages are also authenticated by the SCE-MT Manager or they are authenticated

[3]Apache ODE (Orchestration Director Engine): http://ode.apache.org

[4]WSO2 Business Process Server: http://wso2.com/products/business-process-server/

[5]OW2 Orchestra: http://orchestra.ow2.org

[6]YAWL (Yet Another Workflow Language): http://www.yawlfoundation.org

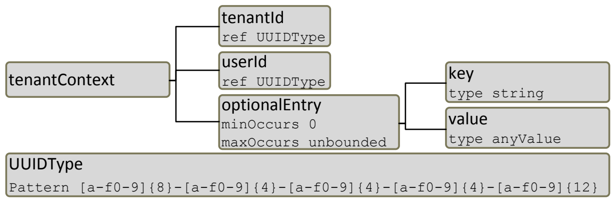


Figure 2. Schema of a Tenant Context to uniquely identify a tenant user [14]

by the SCE internally. Both approaches have their advantages and disadvantages. Since in principle we want to provide fine-grained access permissions on an operation-level, we propose to handle the authentication of management messages inside the SCE. Therefore, the Process/Instance Manager component has to be extended to handle the authentication of incoming calls to the Management Interfaces. This enables the use of context-related data during the authentication like the instance or process model for which a management operation is invoked. If we outsource the authentication to the SCE-MT Manager, we have to query the SCE for the necessary data to handle the authentication. Finally, the new *Configuration Manager* component provides the functionality to configure the SCE and its process models on a tenant basis. It contains all implementation specific code and is provided through a new *Configuration Interface* to the outside. The SCE-MT Manager uses this interface to configure an SCE or one of the deployed process models.

## IV. Realization

Figure 3 shows the overall architecture of *SCE$^{MT}$*, our prototypical realization of the multi-tenant aware SCE architecture discussed in the previous section. The realization approach focused on maximizing the integration and extension of existing multi-tenancy enablement components and the development of new components when deemed necessary. For instance, the SCE-MT Manager modules (see Fig. 1) integrate and reuse two existing components. More specifically, the multi-tenant aware Enterprise Service Bus[7] *ESB$^{MT}$* described in [3] and [15], is used to provide communication isolation for the (process model) services exposed by the SCE. ESB$^{MT}$ is based on the open source ESB implementation Apache ServiceMix[8] (ServiceMix) version 4.3.0. In addition, we use the integration and routing capabilities of the ESB to connect all components (SCE-MT Manager, multi-tenant aware SCE and JBIMulti2) and to hide the complex structure of the integrated system from the users at the same time. Since the SCE also needs a secure administration and management interface, for example to register configuration data, we integrated and extended *JBIMulti2* — a tenant-aware web application for the administration and management of *Java Business Integration* (JBI) environments which is provided with ESB$^{MT}$ [16]. The JBI architecture [17] allows different vendors to "plug in" their components into a standardized infrastructure which enables the decoupling and interoperability between their components on the basis of standards-based messaging.

As in the original ESB$^{MT}$ setup, all registry components in SCE$^{MT}$ are realized based on PostgreSQL[9] version 9.1.1. We added a new Event Registry which stores all event messages emitted by an SCE (e.g. during process instance execution) in a tenant isolated manner. The event data is stored in a separate, distributed database. The existing ConfigurationRegistry and ServiceRegistry are extended to store some new SCE and process model related data. The registries are also used to integrate the SCE-MT Manager with the JBIMulti2 application without having any duplicated data or the need to synchronize data between two separated sets of databases. This is important because the SCE-MT Manager inserts data to the registries later read by JBIMulti2, and vice versa the SCE-MT Manager reads data inserted in the registries by JBIMulti2. As a result, JBIMulti2 and the SCE-MT Manager write and read data to and from the shared registries without the need to send the data to each other. The only required communication between JBIMulti2 and the SCE-MT Manager consists of sending status updates which are simple event messages to inform the SCE-MT Manager that something has changed in the registries. For example, if a tenant registers new configuration data over JBIMulti2, the SCE-MT Manager has to query the data from the Configuration Registry and forward it to all registered SCE instances which provide a process model of the tenant.

JBIMulti2 is extended to support the tenant-based administration and management of SCEs. For this purpose, its Web Service API is extended with new operations which are realized in the corresponding components of the Business Logic layer. For example, a new operation to register configuration data for a process model is added to the Web Service API and its implementation is added to the ConfigurationRegistryManager (see Fig. 3). The new SCE-MT Manager Client enables the required communication with the SCE-MT Manager. For example if new configuration data for a process model are registered, a corresponding message is sent by the client to the SCE-MT Manager over its queue.

The JMSManagementService OSGi[10] Bundle shown in Figure 3 is provided with ESB$^{MT}$ and is responsible for the installation/uninstallation and configuration of Binding Components (BC) and Service Engines (SE). Furthermore, it realizes the tenant aware deployment/undeployment of Service Assemblies (SA) and Service Units (SU) to the BCs and SEs of a JBI environment. We extended the functionality of the JMSManagementService by enabling the feature-based installation/uninstallation of BCs and SEs using the interfaces provided by Apache Karaf[11] which is included in ServiceMix, to simplify the installation of SCE$^{MT}$. The multi-tenant aware SCE realization is based on the open source BPEL Engine Apache ODE version 1.3.5. To provide multi-tenancy support in ODE, we applied the identified adaptations and extensions introduced in Section III to the underlying implementation. The resulting SCE realization is integrated as a JBI Service Engine into ESB$^{MT}$ by using the existing JBI Integration Layer of ODE.

The SCE-MT Manager is realized as an OSGi bundle and is installed to the underlying OSGi platform on which ServiceMix is built on. The way we implement the three layer architecture introduced in Section III is shown on the

[7]ESB$^{MT}$: http://www.iaas.uni-stuttgart.de/esbmt/
[8]Apache ServiceMix: http://servicemix.apache.org/
[9]PostgreSQL: http://www.postgresql.org/
[10]OSGi Alliance: http://www.osgi.org/
[11]Apache Karaf: http://karaf.apache.org/

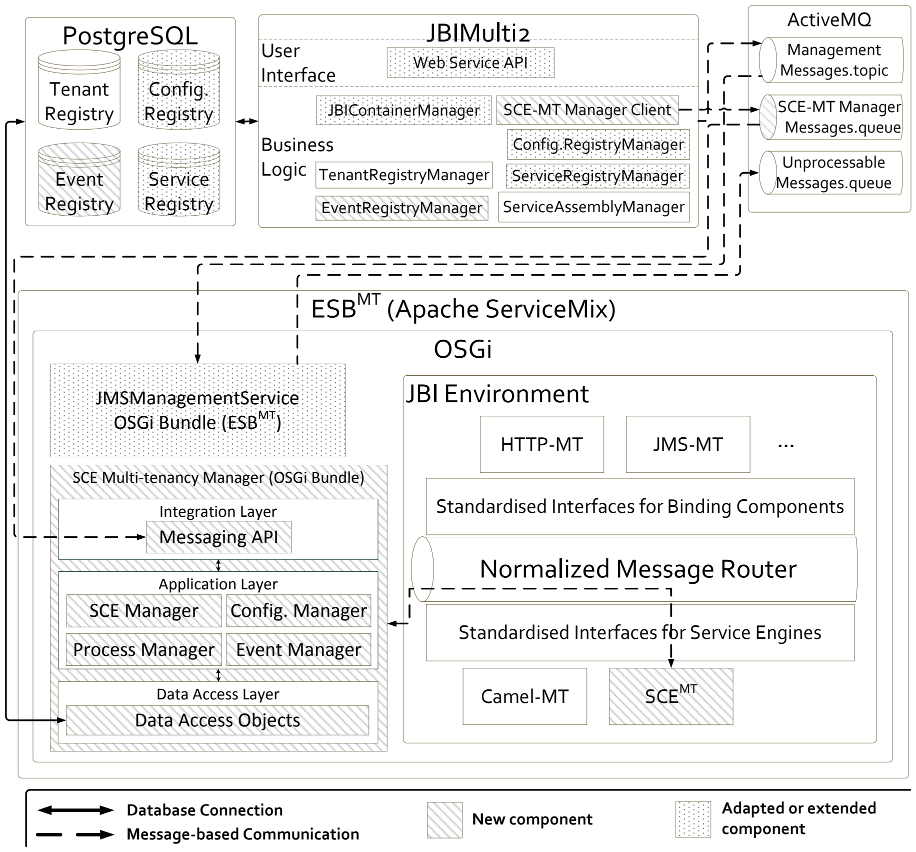


Figure 3. Overall architecture of the SCE$^{MT}$ realization using ESB$^{MT}$ and JBIMulti2

bottom left of Fig. 3. The *Messaging API* is used to enable the communication between the SCE-MT Manager, JBIMulti2 and the connected SCE$^{MT}$ implementations. The *SCE Manager* module provides the required functionality to manage a dynamic changing set of SCE instances and their integration into the ESB during runtime, to support scalability in a future version. The *Process Manager* module is aware of the status of a process model and where it is actually deployed. This can be used in a future version to enable load balancing by deploying a process model to different SCE instances based on the current workload. An additional benefit of this approach is that a tenant does not need to know on which SCE instance(s) its process models are deployed or, on which SCE instance the corresponding process instances are executed. Furthermore, the SCE-MT Manager is responsible for the tenant-isolated storage of all event messages in the new EventRegistry. The *Event Manager* persists the received event messages from all connected SCE instances into the Event Registry to enable their later use. The *Configuration Manager* realizes the configuration of SCE instances and process models. It collects the configuration data from the Configuration Registry and sends these data to the set of SCE instances managed by the SCE-MT Manager. The SCE-MT Manager therefore cooperates with JBIMulti2 and uses the powerful message routing functionality of the ESB by deploying a set of message routes to it.

Further information on the realization of SCE$^{MT}$, including demo videos, is publicly available online[12].

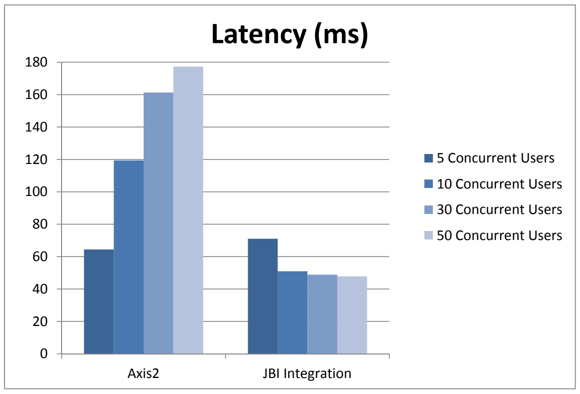
[12]SCE$^{MT}$: http://www.iaas.uni-stuttgart.de/scemt/

## V. Evaluation

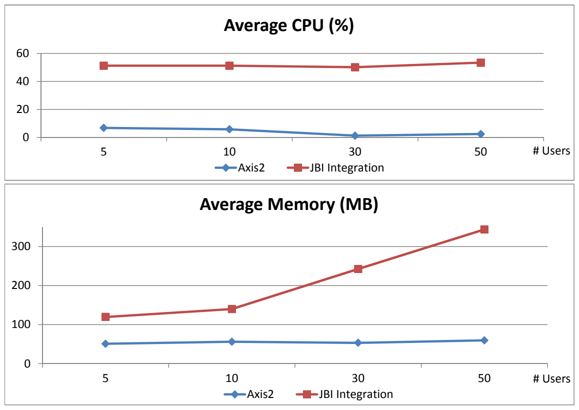
For the purposes of evaluating SCE$^{MT}$, we focus on two different dimensions. The first dimension consists of empirically evaluating two possible realization architecture alternatives: integrating the SCE directly with the ESB as a JBI component (JBI integration), versus the indirect stand-alone deployment and integration through the Apache eXtensible Interaction System v.2 (Axis2[13] integration). The second dimension aims at evaluating the performance variation when introducing multi-tenant communication capabilities at the integration layer, i.e. the support for dynamically creating unique endpoints for a (tenant) user-based process deployment and processing the incoming and outgoing messages in an isolated manner.

Toward this goal, we follow the example of [7], and we measure for the same workload the performance variation perceived by each user when introducing multi-tenancy awareness in relation to the non multi-tenant aware solution. In terms of the evaluation setup, in both cases the SCE is deployed in a virtual machine hosted in an on-premise private Cloud infrastructure with the following configuration: 2 vCPUs AMD Opteron 2,30 GHz, 4GB RAM, and 60GB disk space. Towards emulating a remote access to the processes deployed in the SCE, the load driver is deployed in a separate virtual machine. The generated workload consists of a set of random 1KB SOAP over HTTP messages, which are sent concurrently among the endpoints in different load bursts according to the following function over time: $m(t_i) = w(t_0) + \sum_{i=1}^{5} 2^{i-1} \cdot k \mid$

[13]Apache Axis2: http://axis.apache.org/axis2/java/core/



(a) Latency per endpoint perceived by each tenant user



(b) Resource utilization

Figure 4. Axis2- vs. JBI-based integration of plain ODE with ESB

$k = \{2000\}, w(t_0) = 10240$. For each scenario, a warmup phase $w(t_0)$ of 10240 requests is followed by a set of load bursts that follow an exponential function of base 2, with an initial burst of 2000 requests and a total of 62000 requests. A simple "echo" process model is used in all cases. The latency perceived by each tenant user is measured in milliseconds, and the CPU and Memory utilization in usage ratio and MB, respectively.

With respect to the JBI-based integration of the ESB and SCE (plain ODE) solutions, the evaluation results depicted in Fig. 4a show that the performance is decreased by approximately 5% when concurrently accessing 5 endpoints. However, as the number of concurrently used endpoints are increased, it can be observed that there is a performance improvement of approximately 22%, 43%, and 50% for the scenarios where 10, 30, and 50 endpoints, respectively, are accessed concurrently. Therefore, it can be deduced that in a multi-tenant scenario where an SCE is shared among multiple users, there is a significant performance improvement when the SCE is integrated directly into the ESB as a JBI Service Engine. When looking at resource utilization however (Fig. 4b), it can seen that the JBI-based approach puts additional strain on the computational resources required, which may have an undesired effect as the number of endpoints/concurrent requests increases.

Figure 5 summarizes the results of the multi-tenant versus non multi-tenant performance comparison on an endpoint basis. The defined scenarios consist of evaluating an equal number of configured users for both multi-tenant and non multi-tenant approaches. For example, the evaluation scenario for 10 Endpoints in Fig. 5 consists of comparing the performance of 10 users (10 non multi-tenant endpoints) and two tenants with 5 users per tenant (10 multi-tenant endpoints). The evaluation results depicted in Fig. 5 show that the performance is degraded by approximately 1% when accessing 5 multi-tenant endpoints. However, as the number of tenant-aware and non tenant-aware endpoints are increased, it can be observed that there is no performance degradation, but there actually exists a slight performance improvement of approximately 2%, 4%, and 2.5% for the scenarios where 10, 30, and 50 multi-tenant aware endpoints, respectively, are accessed. It can therefore be deduced that there is no significant performance alteration when introducing multi-tenancy in the SCE$^{\mathrm{MT}}$. These results are consistent with and/or improve on the ones discussed by [7] for their multi-tenant aware SCE implementation.

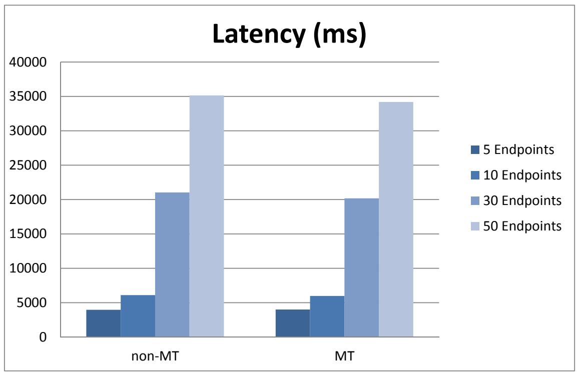


Figure 5. Multi-tenant vs. non multi-tenant endpoint performance comparison

## VI. Related Work

Anstett et al. investigated the execution of BPEL processes in the Cloud based on different delivery models namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [8]. They point out a set of requirements and challenges based on the used delivery model. Furthermore, they identify the need for multi-tenancy on the SaaS level. They do not however describe an architecture or realization approach how multi-tenancy can be supported. Looking specifically at the SaaS model, the work in [6] discusses how a BPMN-based business process execution engine can be made multi-tenant to allow for tenant-based configuration of the application-supporting processes. The discussion however focuses on the modeling aspect of processes and does not provide any architectural insights into how to implement a multi-tenant aware execution engine.

As already mentioned in the previous section, Pathirage et al. introduced a multi-tenant SCE architecture based on the open source ODE BPEL engine [7]. They use the WSO2 Carbon[14] platform to enable multi-tenancy in ODE. WSO2 Carbon is an OSGi-based platform for building scalable, high performance servers. It provides multi-tenancy support by some adaptations to its underlying execution engine Axis2. In [7] the authors reuse the multi-tenancy functionality of

[14]WSO2 Carbon: http://wso2.com/products/carbon

WSO2 Carbon in ODE by adapting the Axis2 integration layer of ODE. An integration layer enables the communication with external services and provides the internal services of ODE to the outside by the use of a service middleware (e.g. Axis2, Enterprise Service Bus). This differs from our approach because the use of Axis2 as multi-tenancy enablement layer and the adaption of ODE's integration layer makes the approach solution specific. Other SCE implementations may use another service middleware or do not offer the ability to provide a set of interchangeable integration layer implementations. We try to separate as much of the multi-tenancy functionality from the SCE internal logic as possible to provide a general and reusable approach, as discussed in Section III. Therefore, we realize communication isolation for the ODE services (process models) at the ESB, by using the corresponding functionality of ESB$^{MT}$. To isolate the process models of different tenants inside ODE, the work in [7] uses an instance of a tenant-aware Process Store for each tenant. In our work, we extended the Process Store of ODE to natively support the isolation of process models on a tenant user basis by associating a process model with the tenant context of its owner.

## VII. Conclusion and Future Work

Enabling multi-tenancy on the level of a Service Composition Engine (SCE) allows for the offering of the engine as a PaaS solution, promoting resource sharing and offering advantages to both service providers and consumers. Toward this goal, in the previous sections we identified a set of functional and non-functional requirements for multi-tenant SCEs including aspects like isolation, configurability, and scalability. Starting with these requirements as the premise we proposed a generic and reusable multi-tenant SCE architecture which decouples the actual engine from the multi-tenancy management concerns. The proposed architecture allows for realization by various SCE solutions and the potential reuse of the management aspect by many SCE instances. We based the prototypical realization of this architecture on Apache ODE, and previous work on enabling multi-tenancy on the level of ESB middleware, resulting in the SCE$^{MT}$ solution. By evaluating SCE$^{MT}$ against a non multi-tenant option in an equivalent scenario we have demonstrated that there are actually small but interesting performance improvements.

Furthermore, when compared with the approach taken by the closest related work [7], we showed that additional performance improvements can be achieved through the direct integration of SCE and ESB. However, a systematic comparison of SCE$^{MT}$ with the solution discussed in [7] is required for validation purposes. Beyond this further evaluation, the design and development of configurable tenant-aware interfaces and GUIs to the SCE$^{MT}$ are also planned for the immediate future. Due to the fact that configurability is one of the most important capabilities of a multi-tenant application from a users point of view, we want to define and classify all possible configuration options — on both the SCE and the process model level — in future work. In addition to that, we plan to elaborate and realize the horizontal scalability of SCE$^{MT}$ based on the ideas introduced in this paper. This provides us the basis to realize the automatic provisioning and de-provisioning of integrated ESB$^{MT}$ with SCE$^{MT}$ instances based on changing workload or performance.

## Acknowledgments

The research leading to these results has received funding from the German Research Foundation (DFG) within the Cluster of Excellence in Simulation Technology (EXC310) and from the EU FP7 600792 project ALLOW Ensembles.

## References

[1] F. Chong and G. Carraro, "Architecture Strategies for Catching the Long Tail," 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479069.aspx

[2] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A Framework for Native Multi-Tenancy Application Development and Management," in *E-Commerce Technology and CEC/EEE 2007*, 2007, pp. 551–558.

[3] S. Strauch, V. Andrikopoulos, F. Leymann, and D. Muhler, "ESB$^{MT}$: Enabling Multi-Tenancy in Enterprise Service Buses," in *Proceedings of CloudCom'12*, 2012, Konferenz-Beitrag, pp. 456–463.

[4] R. Krebs, C. Momm, and S. Kounev, "Architectural Concerns in Multi-tenant SaaS Applications," in *Proceedings of CLOSER'12*, 2012, pp. 426–431.

[5] S. Walraven, E. Truyen, and W. Joosen, "A Middleware Layer for Flexible and Cost-Efficient Multi-tenant Applications," in *Middleware 2011*, ser. Lecture Notes in Computer Science, F. Kon and A.-M. Kermarrec, Eds., 2011, vol. 7049, pp. 370–389.

[6] F. Gey, S. Walraven, D. Van Landuyt, and W. Joosen, "Building a customizable business-process-as-a-service application with current state-of-practice," in *Software Composition, 12th International Conference (SC 2013)*. Springer, 2013, pp. 113–127.

[7] M. Pathirage, S. Perera, I. Kumara, and S. Weerawarana, "A Multi-tenant Architecture for Business Process Executions," in *Web Services (ICWS), 2011 IEEE International Conference on*, 2011, pp. 121–128.

[8] T. Anstett, F. Leymann, R. Mietzner, and S. Strauch, "Towards BPEL in the Cloud: Exploiting Different Delivery Models for the Execution of Business Processes," in *Proceedings of IWCS'09 in conjunction with ICWS'09*, Workshop-Beitrag, pp. 670–677.

[9] F. Chong, G. Carraro, and R. Wolter, "Multi-Tenant Data Architecture," 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479086.aspx

[10] "Organization for the Advancement of Structured Information Standards (OASIS). Web Services Business Process Execution Language Version 2.0. OASIS Standard," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[11] R. Mietzner, T. Unger, R. Titze, and F. Leymann, "Combining Different Multi-Tenancy Patterns in Service-Oriented Applications," in *Proceedings of EDOC'09*, I. C. Society, Ed., 2009, Konferenz-Beitrag, pp. 131–140.

[12] L. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically Scaling Applications in the Cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, 2011.

[13] M. Fowler *et al.*, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[14] S. Strauch, V. Andrikopoulos, S. G. Sáez, and F. Leymann, "ESB$^{MT}$: A Multi-tenant Aware Enterprise Service Bus," *International Journal of Next-Generation Computing*, vol. 4, no. 3, pp. 230–249, November 2013.

[15] S. Strauch, V. Andrikopoulos, S. G. Saéz, and F. Leymann, "Implementation and Evaluation of a Multi-tenant Open-Source ESB," in *Proceedings of ESOCC'13*, ser. Lecture Notes in Computer Science (LNCS), vol. 8135, 2013, pp. 79–93.

[16] S. Strauch, V. Andrikopoulos, S. G. Sáez, F. Leymann, and D. Muhler, "Enabling Tenant-Aware Administration and Management for JBI Environments," in *Proceedings of SOCA'12*, 2012, Konferenz-Beitrag, pp. 206–213.

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

All links were last followed on September 19, 2014.