Development and Evaluation of a Multi-tenant Service Middleware PaaS Solution

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

Development and Evaluation of a Multi-tenant Service Middleware PaaS Solution

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*—In many modern systems, applications or services are realized as compositions of multiple existing services that can be enacted by Service Composition Engines (SCEs), which provide the required functionality to enable their definition and execution. SCEs typically use the capabilities of an Enterprise Service Bus (ESB) which serves as the messaging hub between the composed services aiming at ensuring their integration. Together, an SCE and ESB solution comprise the *service middleware* required for the definition and execution of service-based composite applications. Offering a service middleware solution as a service creates a PaaS offering that allows the service consumers to share the service middleware solution in a multi-tenant manner. However, multi-tenancy support for service middleware solutions remains an open issue. For this purpose, in this work we introduce a general architecture for the realization of a multi-tenant service middleware PaaS solution. This architecture is prototypically realized based on open-source, multi-tenant ESB and SCE solutions. The resulting service middleware provides configurability for service compositions, tenant-aware messaging, and tenant-based administration and management of the SCE and the ESB. We also present an empirical evaluation of the multi-tenant service middleware with focus on the SCE. The results of these experiments show a performance degradation within acceptable limits when scaling the number of tenants and tenant users.**

I. Introduction

The Cloud computing paradigm has become prominent in the last years due to its capability of providing IT resources as commoditised services in an agile manner over the Internet in a on-demand and pay-per use model. Such utility-based delivery of services result in an increase of the advantages that traditional IT infrastructures and their application developers can leverage from, e.g. reduced operational costs and rapid elasticity. Application developers are nowadays capable of selecting among a wide variety of existing SaaS, PaaS, and IaaS Cloud offerings to partially or completely host their applications, based on the administration and management tasks they want to outsource. However, outsourcing an application to the Cloud requires both application and services 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].

Various multi-tenancy definitions have been provided in the research and industry domains in the last years, for example in [3], [4], [6] or [7]. In this work 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*. Our main focus in this research work is on the multi-tenancy support of *service middleware*, that is the sharing possibilities of single *Enterprise Service Bus* (ESB) and *Service Composition Engine* (SCE) instances among multiple consumers, enabling the development, deployment and provisioning of service-based composite applications as supported in the Platform as a Service (PaaS) Cloud delivery model. ESBs and SCEs constitute the main underlying service middleware for deploying and executing service-based composite applications typically defined as *process models* (also known as *workflows* or *workflow models*) [8]. The multi-tenancy capabilities discussed in this work do not only manifest as multiple tenants using the same underlying service middleware 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 [9].

Previous work [10], [11] demonstrated that the concept of Middleware 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 service middleware that separates the service composition aspect from tenant management. By realizing this architecture we create the foundations for a PaaS solution for composite service applications. Toward this goal, we build on previous experience with *cloud enabling* only the ESB as part of a larger PaaS offering [12].

The contributions of this paper can be summarized as follows:

1) A generic, implementation-agnostic architecture for multi-tenant PaaS service middleware.
2) A prototypical realization (SerMid^{MT}) of the proposed architecture composed of a multi-tenant ESB (ESB^{MT}) and SCE (SCE^{MT}).
3) A performance evaluation of the SerMid^{MT} realization with the focus on different sharing configuration scenarios on the level of the SCE.

The rest of this paper is structured as follows: Section II provides a brief background on the Service-oriented Architecture (SOA) paradigm, and some service-based composite applications fundamentals with respect to Cloud computing.

Section III introduces the generic architecture of our multi-tenant service middleware solution (SerMid$^{\text{MT}}$) and its prototypical implementation (ESB$^{\text{MT}}$ with integrated SCE$^{\text{MT}}$). Section IV focuses on evaluating the performance variation among multiple supported process model sharing configurations of SCE$^{\text{MT}}$. Finally, the paper concludes with related work (Section V), and a summary of our findings together with an outlook on future work (Section VI).

II. SOA & Composite Applications

The Service-oriented Architecture (SOA) paradigm has been widely adopted in the domain of software design and provisioning as it establishes the foundations for providing application functionalities as loosely-coupled services which can be used by third-party applications. The SOA foundations are based on the principles of service publication, discovery, selection, and binding [8]. These allow service providers and consumers to expose and find, respectively, functionalities which potentially can be combined resulting in more complex and new service-based *composite applications*. The utilization of services as fundamental elements for application development is mainly addressed by Service-oriented Computing (SOC) [13]. Such service compositions are typically realized by specifying workflow models which are enacted by workflow engines and allow the specification of control and data flows among services.

The connection among Cloud computing and SOC has risen a number of challenges and opportunities, as discussed in [14]. On the one hand, SOC provides the computing of services which enhances the application landscape with dynamic and agile features to rapidly adapt to business changes. On the other hand, Cloud computing provides the services of computing, by adopting the utility service provisioning model on an on-demand and pay-per-use basis. Wei et al. conclude in [14] that challenges that arise when bridging the gap and combining SOC and Cloud actually serve as opportunities, e.g. service discovery can benefit from the usage of different Cloud deployment models by expanding the space of published services.

The Enterprise Service Bus (ESB) and Service Composition Engine (SCE) constitute the core artifacts for the development of modern service-based applications [13]. The ESB serves as the messaging hub between applications aiming at ensuring the integration among them, while the SCE enables the definition and the execution of compositions of multiple existing services (e.g. specified by workflow models). Cloud-enabling such software artifacts with multi-tenancy capabilities for allowing users to deploy and run existing or new composite applications in an isolated manner is therefore a challenging but profitable task that we aim to address in this work.

III. Architecture & Implementation

A. Architecture

Figure 1 provides an overview of the architecture of our multi-tenant PaaS Service Middleware solution *SerMid^{MT}*. In this work we extend the original generic multi-tenant ESB architecture introduced in [3] by the integration of a multi-tenant SCE Instance Cluster at the Resources layer, as shown on the bottom right of Fig. 1. Since our proposal builds on the architecture discussed in [3] (and more extensively in [15]), in the following we focus on the additional or extended

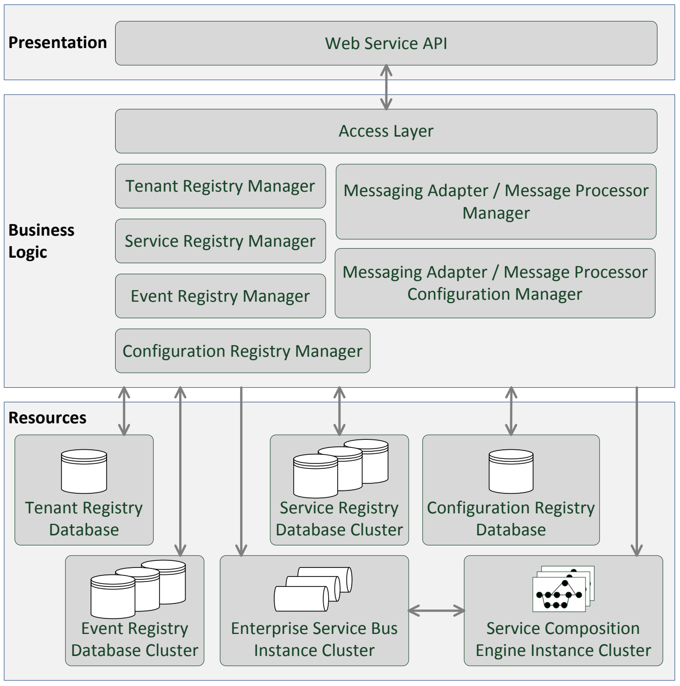

Presentation
Web Service API
Business Logic
Access Layer
Tenant Registry Manager
Service Registry Manager
Event Registry Manager
Configuration Registry Manager
Messaging Adapter / Message Processor Manager
Messaging Adapter / Message Processor Configuration Manager
Resources
Tenant Registry Database
Service Registry Database Cluster
Configuration Registry Database
Event Registry Database Cluster
Enterprise Service Bus Instance Cluster
Service Composition Engine Instance Cluster

Figure 1. Overall architecture of our Multi-tenant PaaS Service Middleware solution (SerMid$^{\text{MT}}$), cf. [3], [15]

components that enable, or are required for the integration of SCE Instances. Therefore, we describe the three layers of the architecture, namely *Presentation layer*, *Business Logic layer* and *Resources layer* in a bottom-up fashion.

Resources layer: The Resources layer consists of a set of registries, an *ESB Instance Cluster* and a *SCE Instance Cluster*. The ESB Instance Cluster is a collection of *ESB Instances* where each instance performs tasks like message routing and transformation that are usually associated with traditional ESB solutions. The components of a generic ESB Instance and their description is out of the scope of this work, but we will provide corresponding details on the level of our prototypical realization later. Interested readers can find additional information in [3], [15] and [16]. The SCE Instance Cluster bundles together multiple *SCE Instances* where each of them is coupled with an ESB Instance of the ESB Instance Cluster as shown in Fig. 1. An SCE Instance enacts service compositions, e.g. specified as workflow models, which for example realize service-based composite applications as introduced in Section II. The number of instances of which the ESB and the SCE Instance Clusters consist can be, in the simplest case, a single ESB Instance coupled with a single SCE Instance which together serve all tenants and their users using a SerMid$^{\text{MT}}$ implementation. Since a single ESB and SCE Instance coupled in a one-to-one manner may cause performance issues, a clustering mechanism similar to the one provided by Apache ServiceMix[1] is recommended for the SerMid$^{\text{MT}}$ architecture. Therefore, the different possibilities to associate ESB and SCE Instances have to be considered. For example, each ESB Instance can be associated with a single SCE Instance or with a collection of n SCE Instances.

In addition to the introduced clusters, the Resources layer contains four different types of registries. The *Service Registry*

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

is extended to store the process models registered with the different SCE Instances in addition to the services of the various ESB Instances as described in [15]. The *Tenant Registry* is used to store data about all tenants and their users as well as associated properties like user names or passwords. The configuration data of all tenants and their users is stored in the *Configuration Registry*. This contains for example the configuration of ESB and SCE Instances or the mapping of permissions and roles to the access control mechanisms offered by the Access Layer component (see Business Logic layer in Fig. 1). In addition to the existing registries, we added a new *Event Registry* which stores all event messages emitted by a running SCE Instance (e.g. during process instance execution) in a tenant isolated manner. Since we want to share the Service Registry and the Event Registry with other PaaS components, we recommend to realize these two registries as database clusters to avoid any performance bottlenecks. Furthermore, all registries have to store the corresponding data in a tenant-isolated manner, for example by the use of one of the approaches (e.g. *shared schema/shared database*) introduced by Chong et al. [17].

Business Logic layer: The Business Logic layer contains an *Access Layer* component which encapsulates authentication and authorization functionality for the tenant users and therefore acts as a multi-tenancy enablement layer [6] based on role-based access control [18]. The Access Layer identifies and authenticates the tenants and their corresponding users once the interaction with the ESB and/or SCE Instances is initiated. In addition, the Access Layer component is responsible for the registration of tenants and users and granting them access to ESB and SCE Instances [3], [19].

Furthermore, as shown in Fig. 1 the Business Logic layer contains a set of Managers which encapsulate the functionality to interact with and manage the components of the Resources layer. All registry managers (*Tenant Registry*, *Configuration Registry*, *Service Registry* and *Event Registry Manager*) provide the business logic which is required to retrieve and store data from and to the corresponding registries in the Resources layer. The *Messaging Adapter/Message Processor Managers* deploy and undeploy corresponding Messaging Adapters and Message Processors in each ESB Instance in the Cluster [3], [19]. Since an SCE is a Message Processor, the Messaging Adapter/Message Processor Managers also deploy and undeploy corresponding SCE Instances in the SCE Instance Cluster and associate them with one or more ESB Instances. The *Configuration Managers* provide the functionality to configure all deployed Messaging Adapters and Message Processors appropriately.

Presentation layer: The Presentation layer contains the Web service API which allows the customization, administration, management, and interaction with the SerMid^{MT} implementation as a PaaS solution. The Web service API offers the functionality provided by the managers of the Business Logic layer for the administration and management of the whole system and also enables the integration and communication of external components and applications [3], [19].

B. Implementation

As discussed in the previous section, the SerMid^{MT} architecture is based on two associated clusters of multi-tenant ESB and SCE Instances. In this section, we introduce our prototypical SerMid^{MT} realization based on the generic architecture shown in Fig. 1 which integrates our multi-tenant aware SCE realization[2] (SCE^{MT}) with an existing ESB^{MT} realization[3] [16]. Figure 2 shows an overview of the SerMid^{MT} realization and all its components based on a single ESB Instance with an integrated single SCE Instance. In the following we introduce briefly all components by going through the figure in a top-down manner. In addition, we describe in more detail how the different components interact with each other.

In [19] a multi-tenant aware administration and management framework was introduced that provides the Presentation and the Business Logic layer of the generic architecture shown in Fig. 1 for Java Business Integration (JBI) [20] environments. Based on that, *JBIMulti2*, a web application was realized which implements the introduced framework, and therefore enables the tenant-aware administration and management of JBI environments. Since JBIMulti2 was designed to only manage ESB Instances that support the JBI specification, we extended JBIMulti2 to support also the tenant-based administration and management of SCE Instances. Therefore, the JBIMulti2 *User Interface* and *Business Logic* layer were extended with new SCE-specific operations. For example, a new operation to register configuration data for a process model is added.

Furthermore, we added a new messaging endpoint to JBI-Multi2 which enables the communication between JBIMulti2 and the *SCE Multi-tenancy Manager* (SCE-MT Manager in Fig. 2). For example, if new configuration data for a process model are registered, a corresponding message is sent through the new messaging channel to the SCE-MT Manager using a queue. The communication between JBIMulti2 and the ESB is based on queue-based messaging since JBIMulti2 is designed to manage clusters of ESB Instances and some management tasks (e.g. installation of JBI components) are long running. For this purpose, in [19] the authors use the open source message broker Apache ActiveMQ[4]. For the purposes of SerMid^{MT}, we added an additional queue (see Fig. 2, *SCE-MT Manager Messages.queue*) to realize a new communication channel between JBIMulti2 and the SCE-MT Manager.

JBIMulti2 uses the set of shared registries introduced in the previous section to handle the administration and management of tenant and user data. The registries are realized based on PostgreSQL version 9.1.1[5]. As already introduced, the *TenantRegistry* is used to store any required tenant information like user names or passwords. The *ServiceRegistry* stores so called JBI *Service Assemblies* (SA) in a tenant-isolated manner. A SA is a JBI-specific packaging format which consists of one or more JBI *Service Units* (SU) providing a collection of component-specific artifacts that can be deployed to any of the components of a JBI environment. All other tenant-related data are stored in the *ConfigurationRegistry*. This contains for example the access rights and roles of all tenants and their users. We added a new *EventRegistry* which stores all event messages emitted by an SCE (e.g. during process instance execution) in a tenant isolated manner. The existing ConfigurationRegistry

[2]SCE^{MT}: http://www.iaas.uni-stuttgart.de/scemt/
[3]ESB^{MT}: http://www.iaas.uni-stuttgart.de/esbmt/
[4]ActiveMQ: http://activemq.apache.org/
[5]PostgreSQL: http://www.postgresql.org/

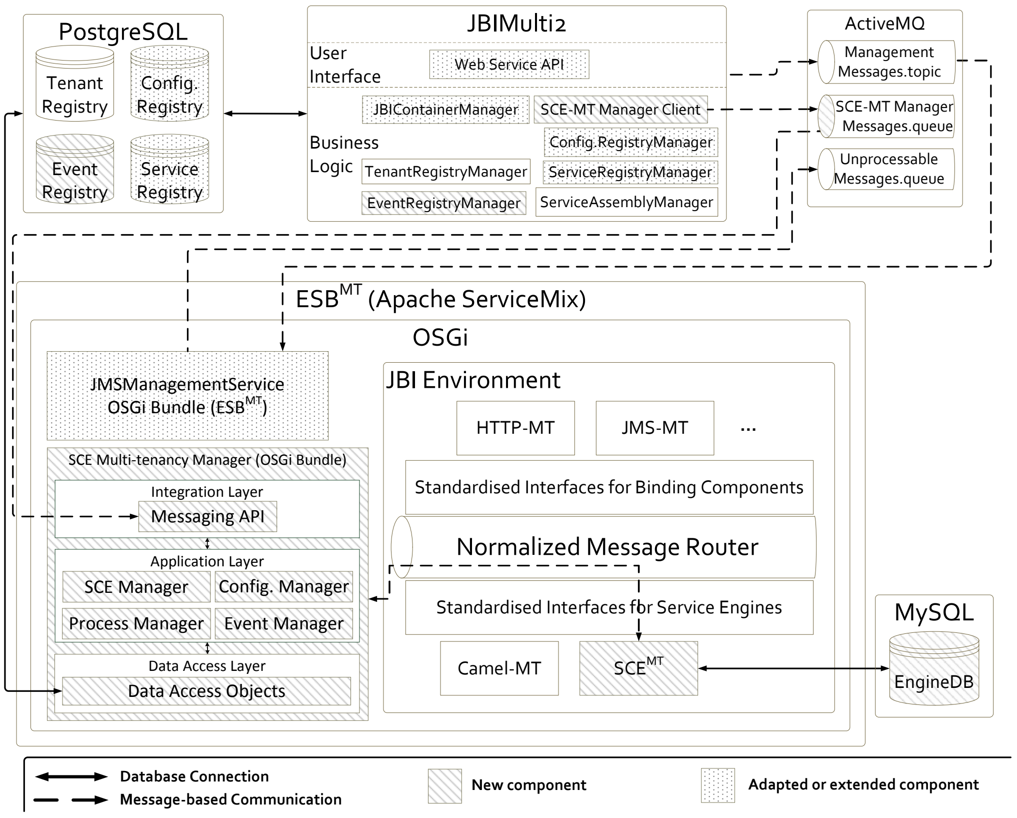


Figure 2. Overview of the prototypical realization of our Multi-tenant PaaS Service Middleware solution based on ESB^{MT} [16]

and ServiceRegistry are extended to store some new SCE and process model related data.

As already mentioned, the ESB^{MT} solution described in [16] is used for the integration of SCE^{MT}. ESB^{MT} is based on the open source ESB solution Apache ServiceMix (ServiceMix) version 4.3.0[6] which is compliant to both the JBI [20] and the OSGi [21] specifications. We use the integration and routing capabilities of ESB^{MT} to connect all components (SCE-MT Manager, SCE^{MT} and JBIMulti2) and to hide the complex structure of the integrated SerMid^{MT} solution from the users at the same time. The *JMSManagementService* belongs to JBIMulti2 and executes all administration and management tasks triggered through JBIMulti2. Therefore, the service consumes all messages published to the *Management Messages.topic* shown in Fig. 2. For example, the service is responsible for the installation/uninstallation and configuration of JBI *Binding Components* (BC) and *Service Engines* (SE). The former realize Messaging Adapters (see Business Logic layer in Fig. 1) and are used to enable the exchange of protocol-specific messages between any connected external service and the JBI environment in a protocol neutral format. The latter realize Message Processors (see Business Logic layer in Fig. 1) and provide advanced message processing functionalities like message transformation, message routing, or even the composition of existing services. For example, an SCE is such a JBI SE which enables the composition of existing services. The *JMSManagementService* realizes the tenant aware deployment/undeployment of SAs and SUs to the BCs and SEs of the JBI environment.

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

The purpose of the *SCE-MT Manager* is quite similar to the one of the JMSManagementService. Instead of providing support for the administration and management of ESB Instances, the SCE-MT Manager enables the management of SCE Instances and as a result of this, acts as the link between JBIMulti2 and the set of running SCE Instances. Its *Messaging API* is used to enable the communication between the SCE-MT Manager, JBIMulti2 and running SCE^{MT} Instances. 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. The *Process Manager* module is aware of the status of all process models and on which SCE Instance they are actually deployed. The *Configuration Manager* realizes the configuration of SCE Instances and process models. For this purpose, it collects the configuration data from the ConfigurationRegistry and sends them to a set of SCE Instances. The SCE-MT Manager cooperates with JBIMulti2 and uses the powerful message routing functionality of the underlying ESB Instance by deploying a set of message routes to it. Whenever an administration or management task is scheduled through JBIMulti2, the collected data (e.g. process model configurations) are persisted to the corresponding registry and then a simple event message is sent to the SCE-MT Manager by JBIMulti2. The SCE-MT Manager uses the information contained in the received messages to get the correct data from the shared registries and trigger the corresponding operation. For example, if a tenant registers new process model configuration data over JBIMulti2, the SCE-MT Manager has to query the data from the ConfigurationRegistry and forward it to all registered SCE Instances. Furthermore, the SCE-MT Manager is responsible for the tenant-isolated storage of all event messages in the EventRegistry. The *Event Manager* persists the received event messages from all connected SCE Instances into the EventRegistry to enable their later use.

The SCE^{MT} realization itself is based on the open source Apache Orchestration Director Engine (ODE) version 1.3.5[7]. Apache ODE supports the enactment of process models specified with the Web Service Business Process Execution Language (WS-BPEL or BPEL) [22]. ODE is shipped with an embedded Apache Derby database, but it is possible to change the underlying database in a configuration file. As shown in Fig. 2 we use a MySQL Server (Community Edition) version 5.6[8] — one of the predefined options of ODE — to provide the required database. Since we can use the functionality of ESB^{MT} to provide communication isolation (separated message exchanges for each tenant user [3]) for all services exposed by the SCE, the SCE does not require a mechanism which enforces that a process model can only be invoked by the tenant user to whom it belongs (owner of the model). But to provide multi-tenancy and configurability support inside ODE, we applied some adaptations and extensions to the underlying implementation. More specifically, the message exchange processor had to be extended to allow tenant-aware communication in both directions. This means that the tenant information associated to any incoming message should be used to identify the sender of the message. Furthermore, the tenant information should be forwarded in all external service invocations. The message exchange processor should be able to handle seamlessly the communication with both non multi-tenant and multi-tenant aware services. In addition to that, the engine database had to be extended to store tenant-specific data (e.g. process models, configurations) in a tenant-aware manner. Our solution is based on the *shared schema, shared database* approach defined in [17] and separates the data of different tenant users based on two new columns (tenantID and userID). To support the tenant-based configuration of the SCE and the process models, we extended the API of ODE with a new configuration interface. A new *Configuration Manager* component implements this interface and by these means provides the functionality to configure the SCE and its process models on a tenant basis. The resulting SCE^{MT} realization is integrated as a JBI SE into the ESB by using the existing JBI Integration Layer of ODE.

Our current prototypical realization of SerMid^{MT} only supports one ESB^{MT} Instance that is coupled with one SCE^{MT} Instance in a one-to-one manner, but we plan to investigate and evaluate different Clusters and association possibilities of ESB^{MT} and SCE^{MT} Instances in the future. For example, a cluster of n ESB^{MT} Instances where each instance is associated with a single SCE^{MT} Instance, or a cluster of m ESB^{MT} Instances where each instance is associated with k SCE^{MT} Instances can be compared. Furthermore, we plan to elaborate and realize horizontal scalability for our SerMid^{MT} prototype and based on that enable elasticity on the level of the two clusters by automatically scaling the number of ESB^{MT} and SCE^{MT} Instances to react dynamically on changing workload or performance.

[7]Apache ODE: http://ode.apache.org

[8]MySQL Community Edition: http://www.mysql.com/products/community/

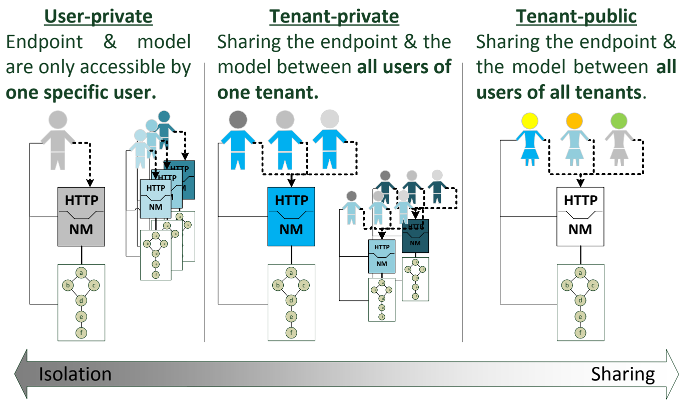

User-private
Endpoint & model are only accessible by one specific user.
Tenant-private
Sharing the endpoint & the model between all users of one tenant.
Tenant-public
Sharing the endpoint & the model between all users of all tenants.
HTTP
NM
Isolation
Sharing

Figure 3. Possible Isolation Types which span a continuum between strict Isolation and unrestricted Sharing

C. Configurability of SCE^{MT}

Our prototypical SCE^{MT} realization already supports a set of configurability options for both the SCE and the enacted process models. For example, tenant administrators are able to configure an SCE Instance by the registration of customized implementations for BPEL *extension activities* on a per-tenant basis. Extension activities are model constructs of BPEL which enable the extension of the BPEL language with new user-defined custom activities that can be used for the definition of process models. To execute these new extension activities, a corresponding runtime implementation has to be registered on the engine side. For this purpose, ODE provides the possibility to register so called *extension bundles* which contain the runtime implementation of one or more BPEL extension activities. We extended this mechanism so that tenant administrators can register such extension bundles in a tenant isolated manner which means only the users of the tenant who registers the bundle are able to use the customized implementation.

On the level of process models, we support configuration options for *process model runtime data* and *process model appearance*. Runtime data can be registered on a tenant user basis for BPEL *variables* and *partner links* of a process model. This enables the customization of (initial) variable values and service endpoint references over runtime data configurations. Figure 3 shows one possibility to configure the *appearance* of a process model, which means how it is visible and accessible from the outside through its service interface. The accessibility of this interface is governed on a per tenant basis by means of constraints. These access constraints enable the owner of a process model to specify a scope with specific visibility and access restrictions and thus only for well-defined groups of tenant users to have access to the corresponding sets of process models. As shown in Fig. 3, we identify three options spanning a continuum between strict isolation (*user-private*) on the one side, and full sharing (*tenant-public*) on the other side for the communication layer of a process model (service interface). Thus they are comparable with the introduced options to introduce multi-tenancy on the database layer by Chong et al.: *separate databases* (strict isolation), *shared database, separate schemas* and *shared database, shared schema* (full sharing) [17]. More specifically, these options are:

Tenant-public: Process models that are configured as *tenant-*

public do not have any access restrictions and therefore are visible to and accessible by all users of all tenants. Any incoming request sent to the service interface of a tenant-public process model is forwarded without any authentication. These process models can be used by any user of any tenant without any restrictions. The fact that a process model is tenant-public does not mean that its instances are also accessible by any user. If a tenant-public process model is instantiated by a tenant user, the instance is directly associated to this user. This secures all tenant-specific instances and their data from any unauthorized entities (Data Isolation). If a tenant-public process model is instantiated without any provided tenant user information, the instance is also tenant-public. In this case, everyone is able to manage these instances via the administration functionality of the SCE. This access constraint enables the provisioning and sharing of process models which provide some useful domain-specific functionality to all tenants and their users while still enforcing Data Isolation on the process instance level. Furthermore, this constraint enables backward compatibility because non multi-tenant aware applications or services are able to send requests to tenant-public process models.

Tenant-private: This constraint isolates a process model on a tenant basis. Therefore the process model is only visible to/ accessible by the users of one specific tenant. The SCE in this cases processes only requests which reference the same tenant as the one associated to the process model to which the request is sent. For example, if the process model is associated to tenant A, only requests of users of tenant A are processed and any other requests are rejected by the SCE. Furthermore, requests without any associated tenant information have also to be rejected. Similar to the tenant-public constraint case, the created instances of a tenant-private model belong to a single tenant user identified by the tenant information contained in the initial request sent to the service interface of a process model.

User-private: Process models which are configured as *user-private* are only visible to/accessible by one specific user. The SCE should only process requests referencing the same tenant and tenant user as the one associated to the invoked process model. Requests without any associated tenant information are rejected by the SCE. The created instances of a user-private model belong to a single tenant user identified by the tenant information contained in the initial request sent to the service interface of a process model. This is the default access constraint which is used when no configuration data is specified for the accessibility of the service interface of a process model.

Further research is planned to identify and define more configuration options on the dimensions of *process model runtime data* and *process model appearance*, and to enable the customization of the logic of a process model by introducing configuration options on the dimension of *process model logic*.

IV. Evaluation

A. Methodology

In previous work we have provided an extended evaluation of the multi-tenancy capabilities of ESB$^{\mathrm{MT}}$ (see for example [15]). For this reason we focus on the evaluation of SerMid$^{\mathrm{MT}}$ at empirically analyzing the performance variation when introducing process model sharing capabilities to SCE$^{\mathrm{MT}}$. As previously discussed, there are several configuration options (isolation types) supported in SCE$^{\mathrm{MT}}$ on both tenant and (tenant) user levels, which must be taken into consideration when analyzing the performance improvement or deterioration among the different levels of supported access constraints. Figure 4 summarizes the scenarios to be considered in the rest of this discussion for purposes of evaluating the effect of multi-tenancy on the level of service middleware.

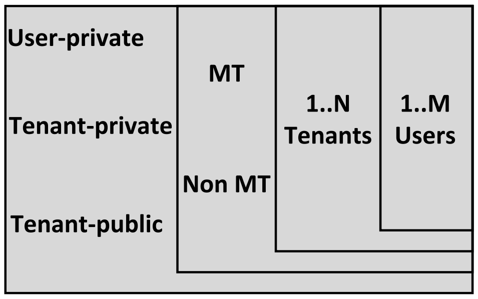


Figure 4. Summary of the experimental evaluation scenarios

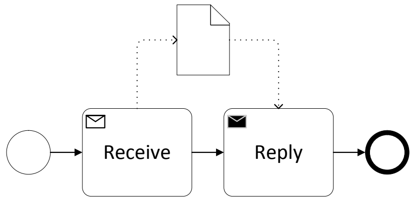


Figure 5. Echo process model used for the evaluation in BPMN2 notation

Based on previous experience from the experiments presented in [15], a workload consisting of randomly generated 1KB SOAP over HTTP requests distributed among two phases was generated for the experiments. The *warm-up phase* consists of a set of 10K messages and is followed by an *experimental phase* comprising a set of 62K requests, both phases following the same function over time:

$$m(t_i) = w(t_0) + \sum_{i=1}^{5} 2^{i-1} \cdot k \mid k = \{2000\}, w(t_0) = 10240$$

Such workload is concurrently sent to the simple back-end echo process model depicted in Fig. 5 which is deployed to a single SCE$^{\mathrm{MT}}$ instance. The echo process consists of a pair of activities (*receive* and *reply*) which use an intermediate process variable to temporarily store and copy the received message content to the response. As these experiments aim at analyzing the impact of introducing multi-tenancy, we must measure the impact from two different perspectives. On the one hand, the latency variation on a per user basis must be analyzed. On the other hand, there exists also an impact from the perspective of the Cloud infrastructure, i.e. the amount of load that each SCE$^{\mathrm{MT}}$ instance is capable of handling when scaling the number of users and requests. For this reason, we also measure the throughput per endpoint supported by each SCE$^{\mathrm{MT}}$ instance.

B. Experimental Setup

The experimental setup complying with the previously discussed evaluation methodology was implemented as follows. The echo process described in Fig. 5 was implemented using the WS-BPEL language [22] and deployed in each SCE^{MT} instance through the extended JBIMulti2 with the corresponding multi-tenancy isolation types proposed in Fig. 4: *user-private*, *tenant-private*, and *tenant-public*. For the non multi-tenant aware scenarios, it was necessary to automate the creation of the endpoint specification files. For this purpose, we used predefined endpoint specification templates and the JET framework[9] to create different non multi-tenant aware endpoint specifications. The scenarios are organized in groups of 1, 5, 10 and 50 users. In the multi-tenant aware scenarios, the users are distributed among 1, 2, 5, and 10 tenants, each tenant comprising 1 or 5 users based on the corresponding scenario (see also Table I). For example, the *MT / User-private / 2 Tenants / 5 Users* and *Non MT / User-private / 10 Users / 10 Endpoints* scenarios both comprise in total 10 users, however distributed among two tenants in the multi-tenant case. The deployed process models and ESB endpoints were allocated differently depending on the isolation type. For the *user-private* isolation type, one process model per user and one unique ESB endpoint are deployed. In the *tenant-private* isolation type, all users of the same tenant share the same process model and ESB endpoint, and therefore one process model and ESB endpoint per tenant are deployed. Finally, in the *tenant-public* isolation type, one shared process model and ESB endpoint for all tenants and their corresponding users are deployed.

Two separate SerMid^{MT} instances (Fig. 6) hosting the multi-tenant and non multi-tenant aware configurations were deployed in the virtual machines VM2 (disabled multi-tenancy support using the backward compatibility feature of ESB and SCE) and VM3 (enabled multi-tenancy support), respectively, both using the following configuration: 2 vCPUs AMD Opteron 2.3 GHz, 4GB RAM, 100GB disk space, and running the Ubuntu 14.04 distribution. SCE^{MT} also required the deployment of the JBIMulti2 components and a MySQL server 5.6 engine database. In order to guarantee an isolated consumption of allocated virtualized resources for each SCE^{MT} instance, the Apache JMeter 2.9[10] was used as the load driver and deployed in a separate virtual machine (VM1 in Fig. 6), with the following configuration: 1 vCPUs AMD Opteron 2,30 GHz, 2GB RAM, 60GB disk space, and running the Ubuntu 14.04 distribution. In terms of the generated workload, random 1KB messages were generated and imported to the load driver. The load driver was configured to concurrently send an equal amount of messages in all scenarios distributed among the users and their corresponding endpoints, i.e. $num\_requests/(num\_users * num\_endpoints)$. For each endpoint the throughput was measured in terms of requests per second (req/s); for each concurrent user we measured the average response time in milliseconds (ms).

C. Experimental Results

As it can be seen in Fig. 7, when comparing the non multi-tenant (*Non MT*) with the multi-tenant (*MT*) scenarios, there exists a detrimental impact to the user-observed performance (response time) when introducing multi-tenancy capabilities

[9]Model To Text - JET: http://www.eclipse.org/modeling/m2t/?project=jet
[10]Apache JMeter: http://jmeter.apache.org/

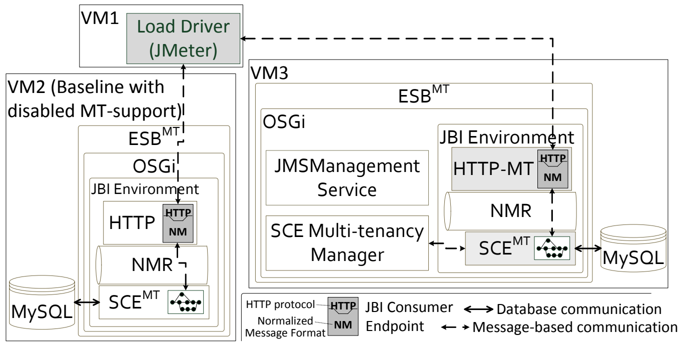


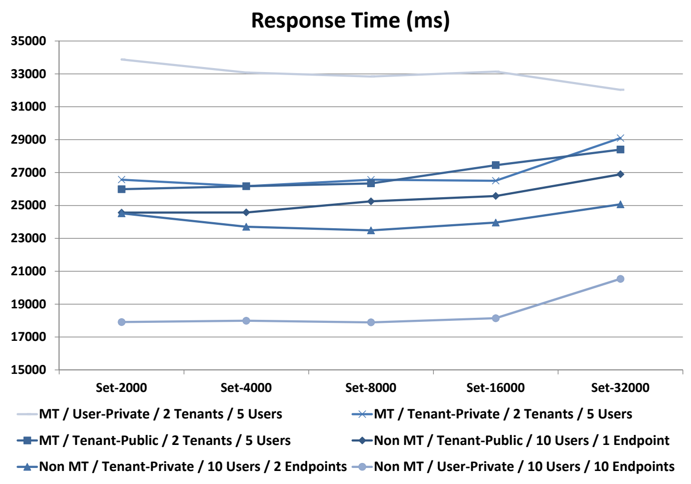
Figure 6. Experimental setup topology

within the SCE[11]. However, this impact varies depending on the isolation type used to configure a process model enacted by the SCE^{MT}. Comparing the user-private scenarios (*Non MT / User-private / 10 Users / 10 Endpoints* vs *MT / User-private / 2 Tenants / 5 Users*) shown in Fig. 7a, the performance is degraded by approximately 44%. When sharing the process model within a tenants' boundaries (tenant-private), the performance is only degraded by approximately 10% and if the process model is fully shared (tenant-public) the degradation reduces to approximately 6%. Overall, the highest performance deterioration can be observed in average for all scenarios related to the user-private isolation type.

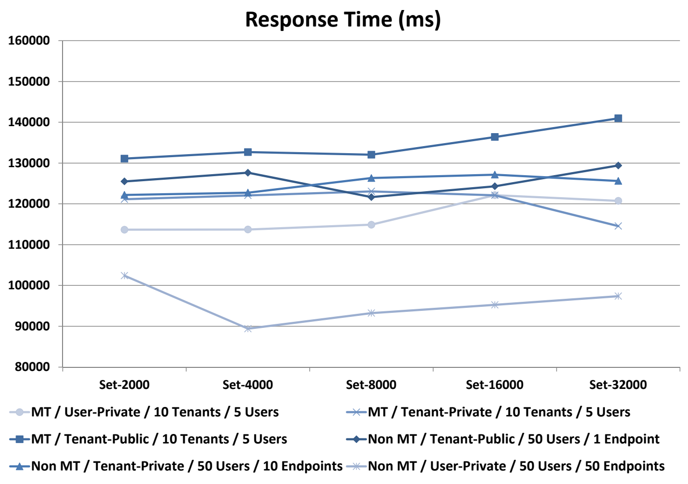
The results also show that the overhead introduced by the authentication mechanisms to realize communication isolation in the SCE^{MT} varies depending on the isolation type. More specifically, the performance degradation variation directly depends on the amount of tenant information that must be processed to authenticate a request at an endpoint. For the user-private isolation type, the authentication mechanism involves the verification of tenant and user credentials, while in the tenant-private type such overhead is reduced since only the tenant credentials have to be validated, and for tenant-public such verification is removed completely. However, the performance degradation perceived by each user tends to be ameliorated when increasing the number of endpoints at the ESB^{MT} and process models in the SCE^{MT}, respectively. For example, comparing the scenarios *MT / Tenant-private / 2 Tenants / 5 Users* (2 endpoints with 5 users per endpoint, Fig. 7a) and *MT / Tenant-private / 10 Tenants / 5 Users* (10 endpoints with 5 users per endpoint, Fig. 7b), there is an average performance degradation of approximately 10% when using 2 endpoints which actually changes to a small performance improvement of approximately 4% when using 10 endpoints.

Furthermore, the response time per user depicted in Fig. 7 shows that there exists a visible latency oscillation among the different scenarios when increasing the number of requests. Such fluctuations result to abrupt beneficial or detrimental performance variations among the different sets of requests constituting the workload. There exists, however, a performance degradation overall trend when increasing the total number of requests concurrently sent per user. With respect to distribut-

[11]*Note:* For purposes of presentation and discussion scoping, Figures 7 and 8 report on the results of only 10 and 50 users, respectively. The measurements for smaller amounts of users (1 and 5) are nevertheless incorporated in later discussion (Table I).

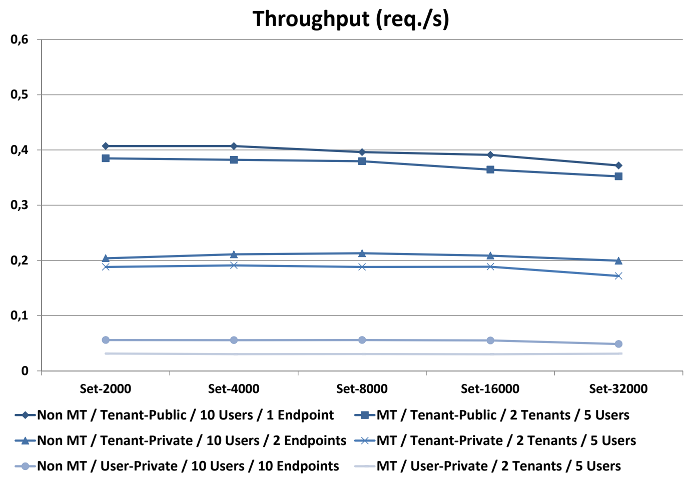


(a) Response time per user for 10 users

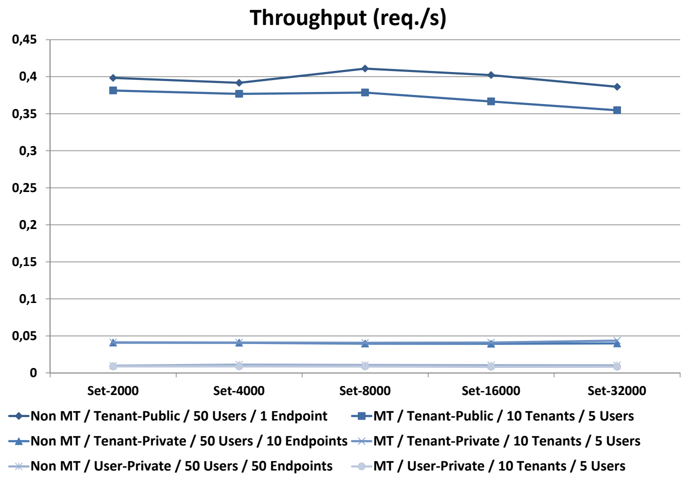


(b) Response time for 50 users

Figure 7. Response time in Milliseconds for all Scenarios



(a) Endpoint throughput for 10 users



(b) Endpoint throughput for 50 users

Figure 8. Throughput in Requests per Second for all Scenarios

ing the load among multiple endpoints, i.e. deploying one independent endpoint per user, we can observe that SerMid^{MT} presents a lower response time for each user compared to the scenarios where multiple users or tenant users share the same endpoint(s). This conclusion is not always applicable to the scenarios comprising a total of 10 users (see Fig. 7a), where the response time highly decreases when user-private multi-tenancy capabilities are introduced (*MT / User-private / 2 Tenants/ 5 Users*). However, when increasing the number of users to a total of 50 users, such degradation is ameliorated with respect to the tenant-public and tenant-private scenarios (see Figure 7b). The impact of enabling multi-tenant aware authentication functionalities for the user-private isolation type highly degrades the response time for each user. However, such difference is decreased on average when scaling to 50 users (Fig. 7b), as the total amount of concurrent requests requiring an user-private authentication mechanism is distributed among the service middleware endpoints.

From the provider's perspective, the performance fluctuation within short time intervals does not show a high variation rate,

Table I. PERFORMANCE VARIATION WHEN ENHANCING THE MIDDLEWARE WITH MULTI-TENANCY SUPPORT

Isolation type	Tenants	Users/Tenant	Endpoints	Average Throughput Difference (req./s)
User-private	1	1	1	-13.74%
	1	5	5	-0.85%
	2	5	10	-44.08%
	10	5	50	-18.45%
Tenant-private	1	1	1	18.79%
	1	5	1	-9.84%
	2	5	2	-10.42%
	10	5	10	3.54%
Tenant-public	1	1	1	-4.66%
	1	5	1	25.04%
	2	5	1	-5.57%
	10	5	1	-6.61%

as the throughput remains relatively steady when increasing the number of requests (see Fig. 8), irrespective of the number of endpoints. Furthermore it can be observed, that the throughput for each endpoint provisioned by SerMid^{MT}, i.e. the number of requests that each endpoint is capable of handling, is also degraded among all isolation type configurations when introducing multi-tenancy support. The average throughout deterioration presented in Table I is highly emphasized in the user-private scenario, while maintained in acceptable margins for the tenant-private and tenant-public isolation type configurations. There exists therefore a clear performance degradation in the SCE^{MT} that depends on the multi-tenant aware authentication mechanisms introduced by each isolation type, signified in Fig. 8 by the existence of three independent bands for each isolation type scenario. In addition to that, there exists also a throughput deterioration with the scaling of the number of endpoints and the distribution of the total workload among them. Such deterioration does not necessarily imply a performance degradation of the system however, as the total number of messages processed per endpoint decreases when distributing the load among further deployed endpoints. Overall, the performance of the SCE^{MT} maintains stable when increasing the number of requests constituting the workload, as the number of requests per second that each endpoint is capable of handling remains steady when increasing the number of concurrent requests. Future experiments aim to investigate further the capacity limitations of the SCE^{MT} when scaling the amount of tenants, users, and concurrent requests.

V. Related Work

Most of the research in the domain of multi-tenancy focuses on enabling multi-tenancy support on the level of SaaS applications, e.g. [6], [23]. There exist only a few existing works that introduce multi-tenancy capabilities for (service) middleware targeted for the PaaS delivery model [7], [10], [11]. All the authors of these works however agree that there is a clear need to introduce multi-tenancy support on the middleware layer. This enables the development of new multi-tenant aware service-based composite applications or to support the cloud-enablement of existing composite applications, respectively, by leveraging the capabilities of the middleware.

In terms of related works to our approach, Walraven et al. [7] introduce a middleware layer that allows the development and execution of multi-tenant applications. Our work however proposes a more generic approach by introducing native multi-tenancy support directly into commonly used middleware components, more precisely into ESB and SCE solutions that are compliant with the JBI specification. In [11] the authors introduce an architecture for a multi-tenant middleware platform, called WSO2 Carbon[12], that enables users to run their services and furthermore provides them an environment to build multi-tenant applications. WSO2 Carbon provides multi-tenancy support by appropriate adaptations to its underlying execution engine Apache Axis2[13]. On top of WSO2 Carbon they support multi-tenancy at the ESB[14] and SCE[15] [10] level. This differs from our approach since the multi-tenancy support is realized by mitigating the tenant administration and communication on the level of the message middleware (Apache Axis2) which makes the approach solution specific. Thus, this method can not be applied to other ESB and SCE solutions. The approach presented in this paper integrates multi-tenancy independently of any implementation specifics of the underlying ESB and SCE solution, as it is based on the JBI specification.

The number of approaches to test and evaluate service middleware solutions steadily grows (e.g. [24]–[27]), but most of them are only focusing on the evaluation of services without taking into consideration the used middleware components and/or the underlying systems [27], [28]. Bianculli et al. developed SOABench, a testbed generation framework which enables the performance benchmark of service middleware solutions [24], [25]. In this work, the authors state that it is important to evaluate the performance of middleware since it provides the runtime environment for the services and service compositions and therefore has an huge impact on the performance of the provisioned services and service compositions, respectively. In [29] the authors support the assertion that the middleware often determines the overall performance and therefore has to be considered. Krebs et al. introduce a performance benchmark for multi-tenant platforms [30]. To the best of our knowledge, there exists no performance evaluation or benchmark that supports the evaluation of multi-tenancy awareness for service middleware so far. Since configurability is an important capability of multi-tenant service middleware, such a benchmark also should be able to evaluate how different configurations of the shared middleware influence the performance of the overall solution.

VI. Conclusion and Future Work

Introducing multi-tenancy on the level of service middleware (i.e. the combination of ESB and SCE solutions required to support composite service-based applications) allows the provisioning of the middleware as a PaaS solution, promoting resource sharing and offering advantages to both service providers and consumers. Towards this goal, we introduced a generic and reusable multi-tenant service middleware architecture which builds on our previous experience with enabling multi-tenancy on the level of an ESB solution. The proposed architecture allows for realization by multiple ESB and SCE solutions and the potential reuse of the management aspect by many ESB and SCE instances. Our prototypical realization of this architecture is based on Apache ServiceMix (ESB) and Apache ODE (SCE), resulting in the SerMid^{MT} solution. SerMid^{MT} also supports (currently) three different multi-tenancy types along the sharing/isolation spectrum. In order to demonstrate the effect of these types we evaluated the performance of SerMid^{MT} by comparing the performance variation among multiple supported process model sharing configurations of SCE^{MT}.

In future work, we plan to investigate the capacity limitations of the proposed SerMid^{MT} solution when scaling the amount of tenants, users, and concurrent requests under a high load. Based on the evaluation results in this work, there is a clear need for capacity planning, in order to find out the workload threshold under which the service middleware is capable of responding in an acceptable manner. Furthermore, our prototypical realization only supports one ESB^{MT} instance that is coupled with one SCE^{MT} instance in an one-to-one

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

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

[14]WSO2 ESB: http://wso2.com/products/enterprise-service-bus/

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

manner. We therefore plan to investigate and evaluate different clusters and association possibilities of ESB^{MT} and SCE^{MT} instances in the future. In relation to this, we will work on introducing horizontal scalability capabilities for our SerMid^{MT} prototype, so that we are able to provide elasticity for the two clusters (ESB and SCE) by automatically scaling the number of ESB^{MT} and SCE^{MT} instances to react dynamically 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 the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*. IEEE, 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*. Springer, 2011, pp. 370–389.

[6] 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.

[7] 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.

[8] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*. IEEE, 2003, pp. 3–12.

[9] 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.

[10] 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.

[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 *Cloud computing (cloud), 2010 ieee 3rd international conference on*. IEEE, 2010, pp. 458–465.

[12] S. Garcia-Gomez, M. Jimenez-Ganan, Y. Taher, C. Momm, F. Junker, J. Biro, A. Menychtas, V. Andrikopoulos, and S. Strauch, "Challenges for the comprehensive management of Cloud Services in a PaaS framework," *Scalable Computing: Practice and Experience*, vol. 13, no. 3, 2012.

[13] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223–255, 2008.

[14] Y. Wei and M. Brian Blake, "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities," *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, 2010.

[15] 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.

[16] ——, "Implementation and Evaluation of a Multi-tenant Open-Source ESB," in *Proceedings of ESOCC'13*, ser. Lecture Notes in Computer Science (LNCS), vol. 8135. Springer Berlin Heidelberg, 2013, pp. 79–93.

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

[18] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based Access Control Models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[19] 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.

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

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

[22] "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

[23] 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.

[24] D. Bianculli, W. Binder, and M. L. Drago, "SOABench: Performance Evaluation of Service-Oriented Middleware Made Easy," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 301–302.

[25] ——, "Automated Performance Assessment for Service-Oriented Middleware," *Faculty of Informatics-University of Lugano, Tech. Rep*, vol. 7, 2009.

[26] X. Li, J. Huai, X. Liu, J. Zeng, and Z. Huang, "SOArMetrics: A toolkit for testing and evaluating SOA middleware," in *Services-I, 2009 World Conference on*. IEEE, 2009, pp. 163–170.

[27] L. Juszczyk and S. Dustdar, *Script-based Generation of Dynamic Testbeds for SOA*. Springer, 2011.

[28] C. Röck and S. Harrer, "Literature Survey of Performance Benchmarking Approaches of BPEL Engines," 2014.

[29] G. Denaro, A. Polini, and W. Emmerich, "Early Performance Testing of Distributed Software Applications," in *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 1. ACM, 2004, pp. 94–103.

[30] R. Krebs, A. Wert, and S. Kounev, "Multi-tenancy Performance Benchmark for Web Application Platforms," in *Web Engineering*. Springer, 2013, pp. 424–438.