Cloud computing, as a concept, promises cost savings to end-users by letting them outsource their non-critical business functions to a third party in pay-as-you-go style. However, to enable economic pay-as-you-go services, we need Cloud middleware that maximizes sharing and support near-zero costs for unused applications. Multi-tenancy, which let multiple tenants (user) to share a single application instance securely, is a key enabler for building such a middleware. On the other hand, Business processes capture Business logic of organizations in an abstract and reusable manner, and hence play a key role in most organizations. This paper presents the design and architecture of a Multi-tenant Workflow engine while discussing in detail potential use cases of such architecture. Primary contributions of this paper are motivating workflow multi-tenancy, and the design and implementation of multi-tenant workflow engine that enables multiple tenants to run their workflows securely within the same workflow engine instance without modifications to the workflows.

**Keywords**-Multi-tenancy, Cloud, SOA, Workflows, Business Processes

## I. INTRODUCTION

By enabling computing power and storage as a utility that is consumed on demand, Cloud Computing has made a significant impression. Many believe that it can bring major savings following the footsteps of electricity, which is one of the major success stories of Utilities [10]. These savings can come in many forms like outsourcing, on demand resource provision, economies of scale, pay-as-you-go etc. Let us look at a few of such benefits.

Most organizations incur significant costs in IT infrastructure while supporting non-key functions of those organizations. However, outsourcing non-key functions and focusing on competitive advantages are considered the guiding principal for organizations. Therefore, most organizations want to outsource their IT functions as much as possible. Enabling outsourcing by moving IT functionality to the cloud is one of the key promises of the Cloud.

Cloud comes in three flavors: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Since SaaS come as customizable turnkey solutions, it is the most promising approach for outsourcing among those three. Examples of such SaaS applications are GMail, Salesforce etc., where end users have to put almost zero operational effort. However, this model does not work for all applications. For specific applications like Customer Relationship Management (CRM) or Email, where behavior is well defined and universally accepted, SaaS applications have greater chances for matching the exact scenario of the users. However, most IT infrastructures have custom behaviors that do not fit with off-the-shelf SaaS applications. To handle such cases, user can go to IaaS, but IaaS can only take-off Hardware maintenance cost, which is anyway a small fraction, and therefore IaaS users have to build, deploy, and run the applications themselves. PaaS provides a middle ground by enabling users to run custom applications in a third party hosting environments. Example of this can be a Web Application, J2EE application, Web Service or a Workflow Engine that runs in a third party hosting environment. There are many kinds of PaaS services available ranging from databases like database.com to messaging services like Amazon Simple Queuing Service (SQS) to Windows Azure and Google AppEngine.

Due to factors like millions of online users, mobile devices, Facebook and viral marketing, the Slashdot effect—that is several factor increase of requests for an application in a small period of time—has become common. Conventional wisdom is to allocate resources for maximum load, but with steep load spikes, such allocations are unrealistic and wasteful. On these settings, Cloud provides an alternative solution by letting the users allocate resources on demand.

However, SaaS, PaaS and above cases have pay-as-you-go model as a key assumption. That is users can ask for resources and use them only when he needs them and should be able to release resources when he does not. If this assumption holds, applications while not being used should cost the user almost nothing. Therefore, to support pay-as-you-go model, the both SaaS or PaaS middleware should be able to support applications owned by many users (we will call them tenants) within the same server while allocating resources on demand.

It is possible to do this through IaaS where one can run a VM per each user. However, often many of the applications and users are not active (in use). If a hosting provider has 10,000 tenants and if only few hundred are in use at a given time, then running a VM for each is a waste. Since booting-up a VM often takes time and does not complete fast enough to serve the first request, keeping VMs in disk and booting VMs on demand is often not practical.

As a solution to this problem, multi-tenancy has been
proposed and is considered a key enabling technology for the cloud. The idea is that the same server instance can support multiple tenants, and that enables hosting organizations to mix and match heavily used and lightly used tenants together, thus enabling them to run the overall infrastructure with much less resources. In summary, with multi-tenancy, the unit of sharing is smaller than a virtual machine, thus enabling more sharing of resources.

SOA plays a major role in Enterprise IT infrastructures. For example, according to 2009 Zap survey [11], about 50% of organizations already use SOA, while more than 25% planned to use it within the next year. Therefore, supporting SOA platform as a service that enables users to outsource their existing applications through a PaaS could make a significant difference to the end user who is trying to outsource their IT functions.

In our earlier works (Azeez et al. [12], [13]), we have focused on data multi-tenancy and execution multi-tenancy in SOA and large scale PaaS architectures respectively. This paper focuses on enabling multi-tenancy for business processes middleware. Business processes (a.k.a. workflows) let users combine many services and define processes, and languages like BPEL provide portable descriptions of business processes. Business processes provide portability. Furthermore, they enable users to abstract out business logic without embedding them within the rest of the code so that those logic can be managed and evolved independently. Moreover, they provide first class support for long running, transactional executions, unlike program languages like java. If compositions are done in programming languages like java, those compositions need to explicitly handle aspects like transactions and process recovery.

We envision a Workflow as a Service (WFaaS), which enables users to host workflows in the cloud. Apart from general workflow advantages, such a hosting service will provide several advantages. For example, this would loosen the deployment and management challenges of workflow engines, and coupled with visual composition editors, such a hosted workflow engine may bring workflows down from enterprises to developers and normal users. Also it can provide a great composition environment for composing services in the Web just like Mashup technologies. However, unlike mashup technologies, workflows are strongly typed and provide much stronger guarantees, and with the availability of many Web APIs, like Amazon, Ebay etc., workflows can handle composition of more critical applications. For example, if there are sets of tasks that are often repeated as a group (e.g. generating summary reports across various accounts, keeping an eye on deals, checking out an item), each can be coded as workflows (process) and reused as a unit.

This paper proposes an architecture and shares our experience in implementing a multi-tenant workflow engine building on top of Apache ODE. Primary contributions of this paper are motivating workflow multi-tenancy, and the design and implementation of multi-tenant workflow engine that enables multiple users to run their workflows securely within the same workflow engine instance without modifications to the workflows.

Rest of the paper is organized as follows. The next section discusses workflow multi-tenancy and challenges in detail, and the following section discusses related works. Section 3 presents the architecture of the multi-tenant workflow engine in detail. Following Section 4 presents a performance evaluation of the architecture. Section 5 discusses design considerations, and finally the Section 6 concludes the paper.

II. MULTI-TENANCY, SOA AND WORKFLOWS

Service Oriented Architecture (SOA) [15] is a paradigm for realizing the business processes, which spans over heterogeneous enterprises. The fundamental concept of the SOA is the service, which typically implements a distinct business function and is available through a well-defined interface. Services are composed to formulate business processes, which address complex business functions of enterprises. Web Services has become the defacto technology for realizing SOA. The applications encapsulating business logic are exposed as web services, and to orchestrate those services, a business process management system (BPMS) is employed. For the past few years, WS-BPEL (Web Services Business Process Execution Language) has been adopted as BPMS by many enterprises, which use Web Services technology for realizing SOA. This work has been motivated by the importance of multi-tenant workflows and the popularity of BPEL as a workflow language.

To understand workflow multi-tenancy, let us consider two examples.

Scientists often compose domain specific tools again and again in their scientific research. With computing (simulations) as the third pillar of science, often those tools are software tools. Scientific workflows have evolved as a field that thrives to provide scientists with infrastructure to compose those tools using workflows, and this model has been the cornerstone of many large-scale science projects (e.g. Linked Environments for Atmospheric Discovery, http://lead.ou.edu/). However, deploying and running such systems is often a challenge and incurs a significant overhead on those projects. A multi-tenant workflow engine would enable many science projects to share a single infrastructure while taking advantage of elasticity and pay-as-you-go properties of the cloud. Furthermore, with this model, a single infrastructure can take care of different science projects each belonging to different domains.

Another interesting use case can be found in the area of Small and Medium size Businesses (SMB). Typically, most of SMBs could not afford to, and may not want to maintain their own infrastructure. A PaaS infrastructure will enable one or few organizations to provide workflow-hosting
facilities for many SMBs. Currently, SMBs often do not use workflow technologies, not because they are not useful, but because the cost and sophistication required to run such technologies are beyond SMBs. Multi-tenant workflow engines can provide a foundation to deliver a cost effective solution to that problem.

Two main goals of multi-tenancy are isolation and sharing, which are contradictory. Thus multi-tenancy design is a tradeoff. As we discussed in the introduction, the primary motivation of multi-tenancy is to enable efficient resource hosting through maximizing sharing. However, at the same time, each user needs complete isolation and often does not want to know that even the existence of the other users. Regarding isolation, it is threefold: data, performance, and execution. Data isolation makes sure one user cannot access data or executable of the other users at any point, and performance isolation avoids negative impacts on the runtime performance of the other users due to a large execution of one user. Execution isolation makes sure one tenants execution flow is independent from the others’.

Workflows are often described through well-known operations, and they cannot make low level calls. Therefore, the problem of securing each others’ invocations are mostly does not exists, and then the isolation comes down to isolating processes’ states.

Furthermore, a multi-tenant setup will inevitably attract more requests, and therefore has to be scalable. Scalability in a multi-tenant setup has several dimensions: that is supporting large number of requests, processes, and tenants.

III. RELATED WORK

Multi-tenancy was by no means a new concept, and the initial work was done in the context of Application Service Providers (ASP) in late 90s and early 2000s. However, the topic received much attention when Cloud computing provided concrete use cases. One of the first discussions that made a significant effect on multi-tenancy research was a maturity model for SaaS applications. Chong et al [4] formulated a four-level SaaS maturity model to be employed to determine the configurability, multi-tenant efficiency, and scalability of a SaaS application. Level 1 is the traditional ASP [1] model where each tenant has a separate SaaS application. In level 2, the configurable instances of the same application are delivered to different tenants. Level 3 brings in multi-tenant efficiency to a SaaS application, that is a single instance serves every tenant, and separate metadata is used to tailor the application to the needs of each tenant. Level 4 extends the level 3 by making a SaaS application scalable where the SaaS vendor runs the multiple instances of a SaaS application and load balances to scale it up. As we shall discuss in the discussion section, the proposed multi-tenant BPS can be scaled as a cluster and it can be placed at the level 4. Furthermore, a business process deployed in a server instance can be configured and monitored through our multi-tenant aware management portal.

Most of the initial work on multi-tenancy was done in data multi-tenancy. Both Jacobs et al. [2] and Chong [3] have outlined three main approaches for data management in a multi-tenant environment: separate databases, shared database with separate schemas, and shared database with shared schemas. They maintained that the lesser the isolation, the higher the scalability and sharing. We use the third model to implement data multi-tenancy as it supports highest sharing.

Several researches have attempted to support customizing a business processes to meet the requirements of different consumers. Mietzner [5] has proposed the concepts of application templates, which can be attached to predefined variation points in a SaaS BPEL process. Consumers can tailor application templates to reflect their needs. Shi et al. [6] has devised a framework for realizing a SaaS business process through the BPEL specification. Their main goal was to facilitate service and process customization at both deployment and runtime and the composition of SaaS services. The aforementioned researches differ from our work as we provide a multi-tenant aware BPEL engine where a tenant can deploy his or her own business processes. Although we do not support the customization of a single business process, using dynamic endpoints and tenant information, our BPEL engine can orchestrate existing SaaS services provided by diverse partners. Furthermore, the same yet customizable business process cannot support the scenarios where the business logic is completely different for different tenants. Our approach would be a practical solution for such scenarios.

Mietzner et al. [7] has formulated three basic patterns related on how services are shared between different tenants: single instance service, single configurable instance service and multiple instances service. In the first pattern, the same service is offered to all the tenants whereas the second pattern considers services having tenant-specific behaviors. The third pattern addresses the scenarios where the service logic is specific for each tenant. They have further discussed the strategies for building a business process using these three service patterns through a BPEL process. The services to be composed should be selected based on tenant information, and it is achieved though passing so-called TenantContext to the service invocation component. Our approach is similar to their approach. Our ODE integration layer, which is responsible for service invocation, has the access to tenant information, and can dispatch service requests to the correct service based on that information. Therefore, we can orchestrate services in each service pattern efficiently.

Cai, Wang, and Zhou [8] have employed the concept of a tenant context in their approach to make an existing web application multi-tenant aware and run in the cloud. They create a tenant context based on a received web request
and associate it to a thread, which carries the tenant context throughout the processing pipeline in the web application, including the remote cloud services such as database servers. Whenever the application performs tenant isolation such as data isolation, the tenant context is used. We can also transparently bring an existing BPEL process into the cloud, as our approach does not mandate any change to the engine that deploys and executes a BPEL process. We only modified the ODE integration layer in supporting multi-tenancy. Moreover, as we also have adhered to the concept of a tenant context, our approach underpins making a shared business process multi-tenant aware in future without undue effort and cost.

Anstett et al. [9] has discussed the challenges in bringing a BPEL processes to the cloud. They have described those challenges with respect to the three cloud service delivery models: IaaS, PaaS and SaaS. Each model poses many requirements on underlying middleware such as the BPEL engine. Providing BPEL through IaaS does not require any change in the BPEL engine. However, the PasS deployment model needs considerable changes in the BPEL engine as it should provide a platform for creating, running and monitoring tenant specific business processes. From security perspective, both configuration data such as BPEL files and runtime data should be encrypted in order to prevent unauthorized disclosure of those data. Furthermore, direct access to the underlying database used by a BPEL engine should be prevented. Our BPEL engine keeps the configuration data in the file system or WSO2 registry per tenant, and can restrict the access for the other tenants using file system security and WSO2 registries security model. The data in the communication can be secured using a HTTPs transport and WS-Security. Providing BPEL through SaaS needs the BPEL engine to be multi-tenant aware as well as to support the same security requirements of the PaaS model. We provide multi-tenancy at the platform level but not at the business process level. Multiple tenants can create and run business processes in the same BPEL engine but cannot customize a single business process. However, a BPEL process running in our BPEL engine can compose SaaS services (partner services), enabling a user to implement some use cases of a customizable business process. For instance, a user needs to select dynamically the partner services based on the tenant information.

In general, multi-tenancy support for a business process has to be evaluated for at least two use cases: shared business process and shared business engine. As discussed in this section, several researches have attempted to support the former use case by making a business process customizable. Our approach primarily has focused on supporting the latter use case. Our BPEL engine provides a multi-tenant aware platform to be reused by different tenants to create, run, and monitor business processes as if each tenant has his or her own BPEL engine.

IV. ODE MULTI-TENANCY ARCHITECTURE

This section describes the multi-tenant architecture of WSO2 Business Process Server (WSO2 BPS). It uses Apache ODE [16] as its BPEL execution run-time, and is built on top of WSO2 carbon platform [14]. The proposed multi-tenant architecture leverages the multi-tenant support of the Carbon architecture (Azeez et. al. [12]).

This section first introduces the Apache ODE architecture and the integration of Apache ODE to WSO2 Carbon platform. Second, we explore the WSO2 Carbon multi-tenancy architecture. Next, we present the multi-tenancy architecture for BPEL process execution and management, which is based WSO2 Carbon multi-tenancy architecture. Finally, we discuss the architectural strategies for scaling multi-tenant WSO2 BPS. Although we present the solution in terms of Apache ODE and Carbon platform, we believe the approach described in this paper can be applied to the other business process execution environments as well.

A. Non-Multi-tenant Architecture

Apache ODE is a Java based, reliable, compact, and embeddable BPEL execution environment, which supports both WS-BPEL 2.0 and BPEL4WS 1.1 (Business Process Execution Language for Web Services). Moreover, it provides an API for managing BPEL process artifacts and running process instances.

ODE consists of four main components: BPEL Compiler, BPEL run-time, Process Store, and Integration Layer.

BPEL compiler validates and converts the BPEL documents into a form that can be interpreted and executed by the BPEL run-time. Process store keeps the compiled BPEL documents. Using ODEs Java Concurrent Objects (JACOB) framework, which is the persistent virtual machine of ODE, the BPEL runtime executes workflows reliably. The runtime provides application-level concurrency, and supports interrupting execution and runtime state persistence.
A persistence store coupled to the BPEL engine keeps runtime data such as the states of process instances, message routing related information, the values of BPEL variables and partner links for each process instance, and the serialized state of the JACOB.

Users can deploy their business processes written in BPEL on Apache ODE. There are two deployment mechanisms: using a web service and a manual process. The web service based approach enables a user to deploy a BPEL process remotely through a service invocation. The manual process involves copying BPEL files into a designated repository folder where a deployment thread periodically polls the folder and deploys the process. When a user deploys a service, ODE first compiles the process given as a BPEL file, then creates an in-memory representation of necessary configurations, and finally stores it in the persistence storage via DAO (Data Access Object) layer.

Pluggable integration layer concept provides the communication infrastructure for the run-time to interact with external services and ODE has two built-in integration layers: JBI (Java Business Integration) based and Axis2 [17] based. Axis2 based integration layer employs Axis2 to deploy a BPEL process as a web service. With the axis2 integration layer, to invoke a business process, a user has to send a message to the web service corresponded to the business process. Upon receiving a message, Axis2 will intercept and inject the message in to the ODE runtime, which takes care of creating necessary process instances or routing the message to an already running instance.

Non-multi-tenant WSO2 Business Process Server (BPS) extends Apache ODE through a WSO2 BPS specific custom integration layer which implements Carbon platform based process store that uses WSO2 Governance Registry to store the configuration data. Furthermore, process management functionalities are exposed through a secure Web service interface. These changes are done to provide a closer integration with Carbon platform, and we refer interested users to Fremantle et. al. [14].

B. WSO2 BPS Multi-tenancy Architecture

Primary goals of WSO2 BPS multi-tenant architecture are isolation of tenants resources and sharing of system resources (process engine) across tenants. We will discuss how architecture pertains to multi-tenancy for workflow, process deployment, message routing, security, process execution, process, and instance management and finally scalability.

Let us start our discussion by understanding multi-tenancy support of Carbon platform. This support is twofold, where it supports multi-tenant data storage and multi-tenant service executions.

Carbon platform provides users with a registry to store their data. As discussed in Azeez et. al. [12], registry is multi-tenant aware and provides isolation across tenants. This is achieved by adding a tenant column for resource table and verifying and filtering results by rewriting queries at the runtime. This step is done at the registry implementation and more details can be found from the aforementioned reference.

Also, as discussed in Azeez et. al. [12], Carbon platform supports execution multi-tenancy by making underline execution engine, Axis2, multi-tenant aware. Axis2 architecture separates its state and execution and Carbon multi-tenancy uses that separation. Axis2 runtime is stateless, and all the state is stored in two hierarchies where Configuration hierarchy holds deployment time configurations and the context hierarchy holds runtime data. Configuration information about axis2 artifacts such as axis2 services is kept within the configuration hierarchy, and Axis2 loads them from deployment artifacts at the startup or through hot deployment. Carbon supports multi-tenancy by having an Axis2 configuration hierarchy and context hierarchy per tenant and sharing a single runtime across all the tenants. Also, in addition to these per tenant AxisConfiguration instances, there is a master AxisConfiguration which will used to dispatch system level requests and to provide the management infrastructure for managing tenants.

As described before, Apache ODE runtime uses a ProcessStore instance to store process artifacts, process definitions, and configurations. In multi-tenant implementation of Apache ODE integration layer, we have introduced a multi-tenant ProcessStore, which we will call the Tenant-aware process store, to achieve isolation of process artifact related resources. Tenant-aware process store handles the persistence of process artifacts as well as lazy loading and unloading of process artifacts to improve resource utilization. ProcessStore uses a tenant specific registry space to store data, and the registry space also provides isolation.

There is a tenant-aware ProcessStore instance for each tenant, and the tenant-aware ProcessStore is responsible for managing versioning and states of all deployed processes in that tenant. In addition, there is a parent ProcessStore, which stores the all tenant aware ProcessStores, and supports lookup those tenant aware process stores by the tenant name.

Let us walk through the process deployment flow to understand how they functions.

There is a repository space assigned for each tenant, and he may deploy a process by authenticating himself and then uploading the process file through a management console, which will copy the file to his designated repository space. Due to security concerns, we do not provide tenants with direct access to the repository file system location.

Once the file is uploaded, a deployer attached to the tenant picks up the process and deploys the process within the tenant-aware ProcessStore. Furthermore, the deployer publishes a web service associated with the process to Axis2. This web service is multi-tenanted, and is deployed only for the given tenant, and this is achieved Axis2 multi-tenancy support as explained before.
As discussed in Azeez et. al.[14], once a message arrives at BPS server, the underline Axis2 multi-tenancy support will dispatch the message to a service that is specific to a tenant. The tenant information is encoded in to the Service End point Address. The service is aware of the BPS, where it converts the message in to a format that can be understood by BPEL runtime and injects the message in to Apache ODE.

Apache ODE will process the message in the same way as in the non-multi-tenanted case. However, when ODE tries to retrieve the process information, he will be talking to the parent ProcessStore, which uses the tenant information embedded in the execution thread to forward all ProcessStore requests to the underline ProcessStore that is specific to this tenant. Therefore, ODE will process the workflow within a context specific to the tenant that is isolated from others. Furthermore, the process versioning concept of Apache ODE ensure that message will not routed to process from another tenant because even you have the same name and namespace for two processes in two tenants, their versions will be different.

Axis2 configuration hierarchy of multi-tenant version supports a event model that notifies the subscribers when a new tenant is added or the tenant information model has changed. The multi-tenant BPS uses this event model to maintain the tenant aware process stores.

To summarize, Multi-tenant BPS uses two mechanisms to achieve isolation. First is to have a Process Store per tenant, which stores data in the tenants registry space that is isolated by data multi-tenancy as described above. All calls by ODE to the ProcessStore are done against the parent ProcessStore, which route those calls to the tenant specific ProcessStore.

Secondly, Multi-tenant BPS uses Axis2 multi-tenancy support to isolate all outside interactions for a tenant, and to determine the current tenant and to validate him. Therefore, multi-tenant WSO2 BPS architecture is mainly based on Axis2 multi-tenancy architecture and ProcessStore for each tenant concept. For each WSO2 BPS node, there will be one BPEL runtime initialized and all the process executions happen at that. In the current architecture BPEL engine is not aware about the multi-tenancy. All the multi-tenancy related logic handled inside the ODE integration layer.

V. PERFORMANCE RESULTS

This section evaluates the overhead of multi-tenancy. To that end, we have deployed the same workflow within both the multi-tenant and the standalone environments and compared their relative performance. The workflow was loaded with 1, 10, 20, 30 clients. Also for each workflow, latency and throughput were measured and Figure 2 depicts the results. Results were collected by running 200 workflows from each client, and for the single client case, we ran 1000 workflows. Workflow engine and clients were placed in two machines each running Intel Core i5 with 8GB with Mac OS, and the backend service was placed in a node with 2 intel i5 CPUs, 4GB memory, and running Ubuntu Linux.

As shown by Figure 2, when loaded, both versions incurred almost the same overhead. We believe that suggests that the multi-tenant version incurs only a negligible overhead. If we considered the architecture, the multi-tenant version execution is identical to the other version except for few lookups to locate the correct tenant related information. Furthermore, although Axis2 multi-tenancy uses java security enforce isolation, those does not come in to play as ODE does not run any of the user provided code. Therefore, we believe above results are justifiable.

VI. DISCUSSION

As discussed in the introduction, one of the primary use cases for Multi-tenant BPS is supporting Workflow as a service (WFaaS) environment that can truly support a pay-as-you-go model. Such an environment will host the workflow platform in the cloud, and customers can securely create and run their own workflows using the platform. Such a model would result in a great reduction of the cost, time, and effort for the customer. By enabling service providers to run multiple workflow engine instances within the same node, multi-tenant BPS can enable greater sharing and almost zero cost for unused services. Such a WFaaS offering will provide several crucial advantages to the end user. For example, by freeing end users from the hassle of deploying and running a workflow infrastructure, WFaaS solutions will lower the barrier of entry to the workflow technologies and coupled with a visual workflow composition environment like XBaya (www.extreme.indiana.edu/xgws/xbaya), workflows will be more accessible to normal developers. Furthermore, organizations may outsource their workflows through WFaaS, thus enabling them to avoid the complications and costs of running the workflow runtime themselves.

Same time, Multi-tenant BPS can be deployed within an organization using a private cloud; in which case, it will provide greater sharing and server consolidation. For example, an organization may choose to support multiple departments through a Multi-tenant BPS server, where it will provide greater sharing with isolation.

In summary, by pushing the unit of sharing to upper layers, Multi-tenant BPS provides greater sharing and enables a pay as you go model, which takes us one more step towards the cloud model.

For any cloud service, security or in other words enforcing isolation is a critical consideration. The isolation needs to be considered in three dimensions: data, execution, and performance.

Multi-tenant BPS achieves data isolation and execution isolation through multi-tenancy support of Carbon platform. For data isolation, we place all the tenant data in the tenants registry space, which is secured through query rewriting by the registry implementation. All external interactions...
are done though a service deployed within the multi-tenant Axis2, and Axis2 verifies and enforces security before anyone invokes those services. Apache ODE runtime is not aware of tenants, but due to the tenant aware ProcessStore, which is backed by the tenants private registry space, each execution is isolated to the tenant under which it is executed. BPEL processes only let users run well-known constructs and ODE does not let them get access to any internal constructs. Therefore, data isolation effectively isolates process execution.

Multi-tenant BPS does not support performance isolation as of now, and currently we depend on monitoring and auditing resources. We are exploring two potential avenues for handling the problem. First solution is to monitor the CPU usage by the thread and killing the threads that are taking too much CPU. The other option is to monitor the CPU usage and change the priority of the workflow accordingly. Since ODE uses a queue based model, changing priority will solve the performance isolation to a great extent.

Multi-tenant BPS aggregates many users together and therefore tends to receive larger number of requests. Consequently, scalability is of a paramount importance. Scalability of Multi-tenant BPS has several dimensions where it should scale for many tenants, many workflows, and large workflows. To handle the first two cases, we use the following architecture.

We run multiple BPS nodes in a cluster, where tenants are partitioned across few BPS nodes. For example, tenants t1, t2, may be in the node one and tenants t2, t3 can be in the node two. A load balancer that is aware of both tenants and processes sits in front of the BPS cluster and routes messages to a cluster node that supports the target tenant for the message. Furthermore, the load balancer routes all messages belong to the same process instances always to the same node, hence workflows do not need to synchronize data across the BPS instances. Consequently, the BPS cluster uses a shared nothing architecture and can scale freely. We can scale up the load balancer by adding multiple load balancers that have identical routing tables together with a DNS based round-robin model to load balance across those load balancers, which will in turn load balance BPS nodes in the cluster.

Since the above model always executes each process instance in a single node, it cannot scale to handle very large workflows that might not fit within a single node. However, all workflows we encounter are of moderate size, and we do not scale for large workflows as of now. On the other hand, Apache ODE execution model is implemented based on the work item concept where the work items are stored as jobs and executed through a job scheduler. It is possible to make this job scheduler distributed and expand execution of large processes across several WSO2 BPS nodes.

Another interesting aspect of Multi-tenancy is super-tenant (multi-customer) workflows. These are applications available for all the tenants, and the tenant administrators can choose and enable the application within their tenant. For example, Google Apps Market Place features many applications (e.g. Tripit) that can be added by tenants. Such a model can give rise to a market place (WorkflowStore) like Apple Store where users can sell workflows to be used by other users. For example, one can represent a new algorithm as a workflow and sell it as a service through WorkflowStore. While supporting this, tenants can either run the workflows themselves or do a call to a workflow deployed in the offering tenant who will execute it within the same tenant. We can support the first model by either copying the super-tenant workflows to the tenant repository or changing the tenant repository implementation to also include super tenant workflows. Supporting the second model is a matter of providing those users with necessary permissions, or in other words, supporting security federation. Another interesting aspect is supporting users to customize the workflow. We plan to explore the topic as future work.

VII. CONCLUSION

As we discussed in the introduction, Cloud has provided several tangible possibilities for cost savings by providing computing resources as a utility, on demand, in a pay-as-you-go model. But a typical cloud deployment likely to have many inactive or less active users, and one of the main
promises of the pay-as-you-go model is zero or very small cost for users while they are not using the cloud. However, as we discussed in the introduction, supporting this model through VM level sharing would incur long delays, which often are not acceptable for PaaS and SaaS applications. Multi-tenancy has been proposed as a way to achieve better resource sharing and to provide almost zero cost when unused.

Furthermore, we observed that SaaS provides the best cost savings but often only works for some classes of applications. Therefore, PaaS has become an attractive medium to maximize benefits of the Cloud. SOA is a key component of the Enterprise architecture, and supporting SOA as a Service would enable users to migrate their applications to the Cloud with minimal changes. This paper focuses on a multi-tenant architecture for Business process executions, which we call Multi-tenant BPS.

Primary contributions of this paper are motivating workflow multi-tenancy, and the design and architecture of a multi-tenant workflow engine that enables multiple users to run their workflows securely within the same workflow engine instance without modifications to the workflows. The Multi-tenant architecture is implemented as an extension to Apache ODE leveraging multi-tenant capabilities of the Carbon platform. The architecture enable users to deploy their existing workforces as is in a multi-tenant setting without any changes. Therefore, we believe migration would effortless. The multi-tenant architecture is a key enabler for Workflow as a Service offerings, which would enable users to outsource their workflow executions to the cloud.

We discussed the design and implementation of the Multi-tenant BPS and also presented a performance evaluation of the architecture. We observed that the architecture incur almost no overhead, which we believe is a promising result. Furthermore, we discussed the clustering architecture for the multi-tenanted workflow engine and observed that we can partition workflow execution by tenants and by processes with out any need for sharing data across. Therefore, we can scale the architecture using the Shared-Nothing model.

Finally we discussed usecases for Multi-tenant BPS and observed that it can be used as a key component of SOA PaaS offerings. Furthermore, it can be also used within organizations with the private cloud to maximize resource sharing among different units. Moreover, it can be used to offer Workflow as a service where users can use it with a visual composition tool to compose different API calls to create Web API level mashups. Currently multi-tenant BPS is deployed as a part of WSO2 Stratos PaaS platform and available through http://cloud.wso2.com.

REFERENCES