Building a Customizable Business-Process-as-a-Service Application with Current State-of-Practice

Fatih Gey, Stefan Walraven, Dimitri Van Landuyt, and Wouter Joosen

iMinds-DistriNet, KU Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium
{fatih.gey, stefan.walraven, dimitri.vanlanduyt, wouter.joosen}@cs.kuleuven.be

Abstract. Application-level multi-tenancy is an increasingly prominent architectural pattern in Software-as-a-Service (SaaS) applications that enables multiple tenants (customers) to share common application functionality and resources among each other. This has the disadvantage that multi-tenant applications are often limited in terms of customizability: one application should fit the needs of all customers.
In this paper, we present our experiences with developing a multi-tenant SaaS document processing system using current state-of-practice workflow technologies from the JBoss family. We specifically focus on the customizability w.r.t. the different tenant-specific requirements, and the manageability of the tenant-specific customizations.
Our main experiences are threefold: (i) we were insufficiently able to modularize the activities and compositions that constitute the document processing workflow, (ii) we lacked support for describing tenant-level variations independently, and (iii) the workflow engine we employed is too centralized in terms of control, which limits resilience and thereby endangers scalability of the document processing application.

Keywords: Software-as-a-Service, Business Process, jBPM, Multi-tenancy, Customization, Document Processing

1 Introduction

Application-level multi-tenancy is an increasingly prominent architectural pattern in Software-as-a-Service (SaaS) applications. Different tenants are served simultaneously from the same run-time instance of the application while features of the application remain logically separated on a per-tenant basis. This suits best for applications where all potential tenants have highly similar (non-)functional requirements for the application. In case the tenant requirements differ slightly (or even profoundly), customization is required as an architectural feature to the SaaS application to facilitate efficient incorporation and management of tenant-specific requirements.

In the context of an ongoing project [1], we analysed a multi-tenant SaaS application for document processing of an industrial partner that currently serves a

W. Binder, E. Bodden, and W. Löwe (Eds.): SC 2013, LNCS 8088, pp. 113–127, 2013.
© IFIP International Federation for Information Processing 2013

large amount of companies. Although their tenants differ in terms of specific requirements, they do share the common requirement of processing large volumes of documents and data through a multi-step processing scheme, e.g. after document generation additional processing steps such as signing may be required. In summary, the document processing represents a system for workflow-centric processing of batch jobs.

The application that is currently in use follows an ad-hoc software management approach: application functionality for document processing is available as reusable library functions. For each tenant, an individual application is created and executed. This approach suffers from being error-prone and not efficiently manageable (e.g. in case of changes to the document processing system).

In this paper, we present our experiences with the development of a customizable multi-tenant SaaS application that is configured by the tenant, whose workflow is run on top of JBoss' jBPM [2] and whose document processing facilities are modelled as Web services on top of JBoss AS7 [3]. As variability modelling and service variability is out of scope of this paper, we focus on the business process modelling (BPM) and execution aspects of our document processing system.

By showing that batch-oriented business processes with custom requirements (of a particular application domain) can be run as a SaaS application with manageable efforts on state-of-practice tools, we encourage companies with similar settings to migrate their workflow-driven application to a cloud platform. Research has already been performed in adjacent fields, such as feature-oriented domain analysis [4] (variability analysis) and multi-tenant customization [5] (middleware to enable variability in services), but has not been focussing on (practical) studies enlightening the business process aspect of customizable SaaS applications. We believe that this is one reason for low usage of the Cloud paradigm for companies of aforementioned types. Our concept envisions a set of pre-designed workflows provided by the SaaS application developer from which a tenant can simply select and configure the most suitable one and use the business processes execution on-demand as a Service (BPaaS). Simultaneously, by tackling the efficient manageability aspect of a workflow-driven, customizable multi-tenant SaaS application, we also motivate to operate the provider-side of such an application.

This paper is organized as follows: Section 2 introduces the document processing application and motivates the requirements of interest. Section 3 discusses our implementation while Section 4 provides an in-depth discussion of our key decisions and experiences from which we distil challenges and drawbacks that are relevant beyond the scope of this single implementation project. Section 5 discusses related work, and we conclude this paper in Section 6.

2 Problem Illustration and Motivation

In this section we first describe the document processing system that is currently in use by our industrial partner. Then, we highlight the drawbacks in terms

of manageability and summerize requirements for our implementation of the document processing system as a customizable multi-tenant SaaS application.

2.1 Document Processing System

The system of interest in this paper is that of a Belgian SaaS provider. This company, hereafter referred-to as Document Processor (DP), provides a platform for generating, signing, printing, delivering, storing, and archiving documents, and they offer these B2B facilities as a Software-as-a-Service (SaaS) application to their customers (tenants). As a result of adhering to multi-tenancy at the SaaS paradigm, the document processing system is difficult to customize: the benefits of scale inherent to SaaS rely on the fact that the same application can be reused by many different tenants. Nonetheless, as the processing facilities are of relevance to a wide range of companies in very different application domains, several tenant-level variabilities and customizations exist. To illustrate, we present two such tenant companies and their document processing requirements.

$TenantA$ is a temporary employment agency which requires printed payslips to be delivered to its employees. It provides the raw data to DP, with metadata attached to each document. $TenantB$ is in the financial business and uses the document processing facilities for generating invoices and distributing them to their customers (end users). $TenantB$ provides only raw data as input and requires its custom layout to be applied to the documents generated (for branding purposes) and the distribution of documents depending on the end-user's preference (email and printed paper).

2.2 Challenges

In their current document processing offering, the document processing provider uses a set of functional libraries to realize the superset of document processing activities. As each document is processed by a sequence of these activities, the processing logic is realized in the form of Java code in which these libraries are called sequentially. To realize a tenant-specific customization of the application, a variation of this processing logic is created manually – by copy-pasting the existing Java code and making the tenant variations manually.

This approach has several obvious drawbacks: (1) There is no systematic reuse of customization knowledge, and techniques such as copy-paste are error-prone. Moreover, the management complexity of these different variants grows exponentially with the number of supported tenant variations. (2) Because the workflow logic is currently written in a programming language (Java), application administrators are required to be developers skilled in that language in order to set-up new tenants. (3) Whenever the libraries change, these changes ripple through to the different workflow definitions: they needs to be changed manually which does not scale for large number of tenants.

In this paper, we report on our experiences of migrating the existing document processing application to state-of-practice workflow processing techniques (from the JBoss family), and this obviously in the context of multi-tenant SaaS

applications. Specifically, we focused on addressing the key requirements listed below:

- **Manageability of Variations.** In order to remain competitive, the time-to-market of a specific tenant variant is of crucial importance. Therefore, adding new tenants (tenant acquisition), changing tenant configurations, extending tenant variabilities, or modifying the interfaces to the document processing activities have to be more efficient, and the configuration process itself less error-prone.
  Furthermore, the tooling should be suitable to be used by business analysts and domain experts, rather than by developers and programmers. As document processing workflows consist of a set of pre-existing activities out of which a particular sequence is defined, the tool is not required to nor should provide the expressiveness of a general-purpose programming language.
- **Resilience of Workflow Execution.** It is especially important for SaaS applications in a distributed setup, such as for our document processing system, that the workflow execution is resilient against failures of remote services, as failure of nodes is likely, and Service Level Agreements (SLAs) in SaaS contexts tend to approach maximum utilization of resources so that such failures may have severe impact on the fulfilment of SLAs.

Section 3 discusses the relevant implementation decisions. Subsequently, Section 4 provides an in-depth discussion of our main experiences and findings.

3 Implementation

In this section, we describe the implementation of a customizable multi-tenant SaaS application for document processing that we have built to address the challenges discussed in Section 2.1. In line with the scope of this paper, we focus on the business process modelling aspect that we have implemented using the business modelling language *Business Process Modelling and Notation (BPMN)* and its state-of-practice execution engine and modelling tool jBPM.

First we define – for the sake of clarity – the common terminology that is used in the context of jBPM and that we use to describe our experience with the implementation. We then give a short overview of the end-to-end application before discussing its business processing modelling aspects.

3.1 Terminology

A *workflow* is a sequence of (business) *activities*. The persistent artifact in which a workflow is defined, e.g. using BPMN, is a *process definition*. For each execution of a workflow a run-time instance of the process definition, a so called *process instance* is created. The implementation of an activity is called a *task*. Each process instance can have *process instance variables* that may have been set at process instance's creation time and are accessible from within the tasks.

3.2 Overview of the End-to-End Application

The application we illustrate in this section processes document (processing) jobs which are uploaded to the system. Such a job contains a set of input files (either ready-to-distribute documents or raw data for document generation), meta-data for each input-file, and a tenant-ID.

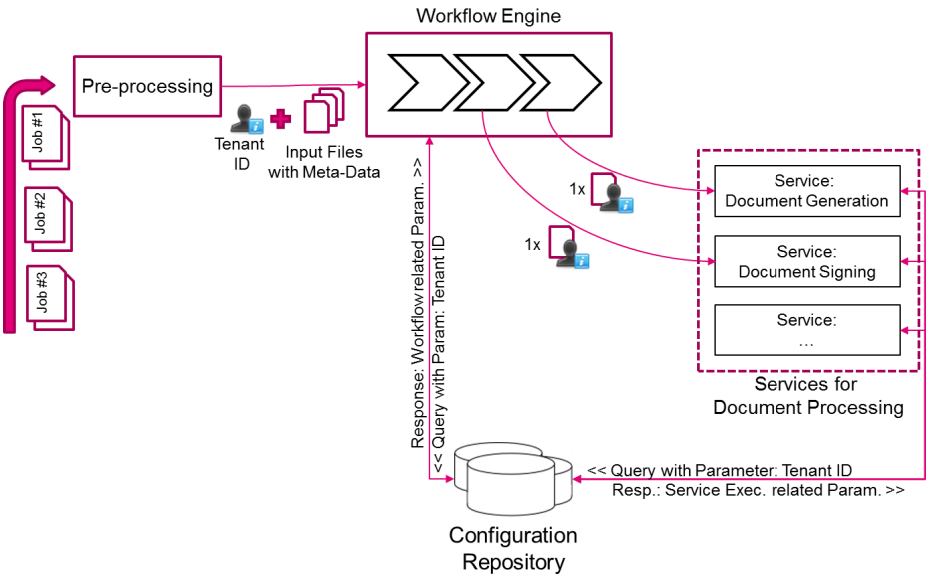

Fig. 1. High-level architecture of the end-to-end application

Figure 1 shows the overall architecture of the document processing SaaS application. A document job is received at the pre-processing component and passed on to the workflow engine. The workflow engine uses the tenant-ID of that job to fetch the corresponding workflow-related tenant-specific configuration from the central configuration repository. For example, tenant A's workflow is configured such that no document generation is executed, but that the input documents should be printed and distributed via postal mail. For each activity of the on-going workflow, e.g. document distribution for tenant A, the corresponding service is called.

Each service fetches its configuration using the tenant-ID, e.g. the template to use for printing tenant A's documents.

3.3 Business Process Modelling

The main business logic of the document processing case is modelled in two different workflows: the outer workflow is represented in Figure 2 and embeds

the inner workflow represented in Figure 3. The outer workflow iterates over the input files of the uploaded document job and invokes the inner workflow for each individual document.

Starting with the Start Event (circle labelled with "S" at the left-top side of Figure 3), the graph depicts the sequence of workflow activities that is run. Activities that are optional have a parallel edge connecting their predecessor with the successors of that activity. Alternative activities are placed as parallel paths to the actual activity. For both variations, `XOR`-typed gateways are used which will proceed the workflow by selecting one of the available outgoing edges depending on dynamically-evaluated code, which we call *switch code*. A switch code may be written in Java code or Drools Rules (a domain-specific language of JBoss for workflows). In addition, gateways of type `AND` are used which result in executing all out-going paths.

As mentioned in Section 3.2, each of the activities of this process definition, when triggered, executes a service call to the document processing services passing the document that is currently being processed and its related data.

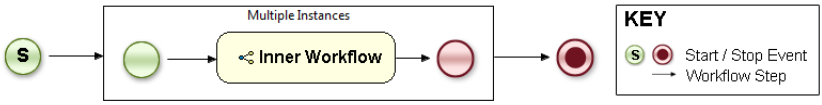

Fig. 2. Process Definition for Document Processing: Outer Workflow

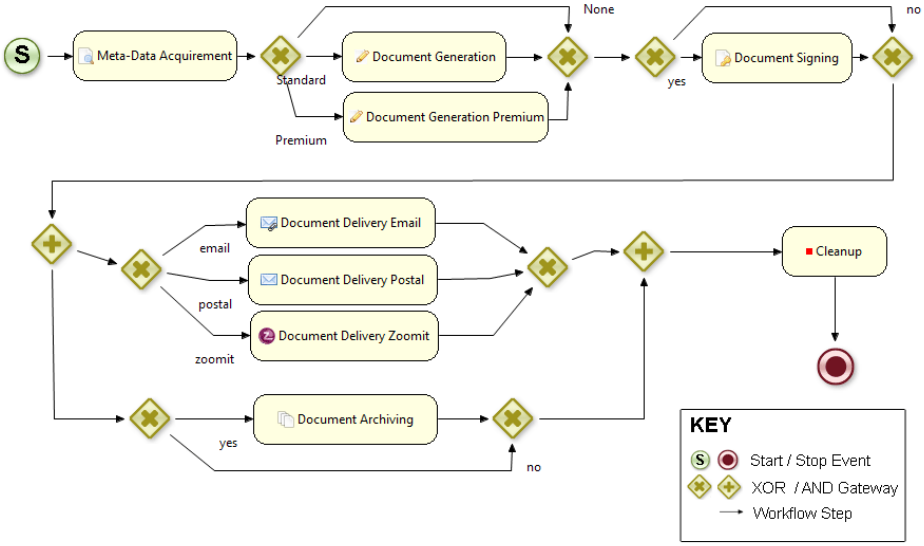

Fig. 3. Process Definition for Document Processing: Inner Workflow

Variability Modelling. As mentioned earlier, `XOR` gateways are used to express optional or alternative activities in the document processing workflow. More specifically, we use them to express tenant-specific variability in the sense that the inner workflow, as shown in Figure 3, depicts the workflow with all tenant-specific variants *included*. Hence, we call this type of artifact a *multi-tenant process definition*.

At the instantiation of a workflow execution (process instance), the tenant-specific parameters required to customize the workflow's multi-tenant process definition at run time are fetched. For tenant A, these parameters are *no document generation* and *delivery method is postal*, and for tenant B the parameters are *document generation method using custom templates* and *delivery method is postal* (or *delivery method is e-mail*, respectively). Those parameters are set as process instance variables which are accessible to all gateways and tasks within the multi-tenant process definition. The aforementioned `XOR` gateways read these parameters and select the tenant-specific options accordingly. For example, for tenant A, the gateway selecting between the delivery methods will read instance variable for tenant A and will select the *postal* option. As a result, workflow executions for each tenant follow only one path from start to finish of a workflow. After a workflow for tenant A has been initialized, no other delivery method than postal delivery is available for the duration of that process instance.

Passing Variables between Tasks and into an Iteration Activity. The BPMN language provides two options for a task to retrieve data: (1) Process instance variables which are variables in the scope of a process instance, i.e. each task can access those, and (2) parameter mapping, a mechanism that can map process instance variables to input parameters of a task or output data from a task to a process instance variable.

Using option 1, all processes would be able to read and write to a scoped global variable space which would limit modularity. In the current version of BPMN, option 2 is limited to map a variable's content to another variable or vice versa, without providing the ability to map a member of a variable (assuming it is an object) to another variable. As a result, using options 2, tasks are expected to know which data are required their successor tasks in order to provide those in separate variables. That is, in case the set of tasks changes, programmatic changes on the tasks are becoming necessary in order to reflect on the changed set of output variables that this task has to fill in.

Option 2 also causes another issue when considering a workflow with iterations. As described earlier (cf. Section 3.3), the outer workflow of the document processing system is triggered with a set of input documents (and per-document meta-data) over which it iterates, calling the inner workflow for each document. A consequence of that architecture is that the document processing system requires the iteration activity of the outer workflow to pass multiple variables (document and meta-data) to the inner workflow, which is not supported by the current version of BPMN. Using option 2, the members of that single variable that is passed into each iteration cannot be mapped to the according parameters within the inner workflow.

As a workaround, we decided to introduce a composite data structure that provides (i) members to store the input document *and* meta-data, and (ii) read- and write-access for additional information. The composite data structure is stored as process instance variable and is accessible by each task (as in option 1). It is used to pass partial results, e.g. the document in question in its (potentially intermediate) current state after each activity, among other book-keeping information, such as a list of so-far completed activities, between tasks. For iterations, it is used to reduce the amount of variables that are required within the inner workflow to one.

4 Discussion

This section discusses our experience and main findings with our implementation (presented in the previous sections) with regard to the requirements we set up for our document processing application (cf. Section 2.2).

4.1 Manageability of Variations

Our findings related to manageability are twofold: (1) Using BPMN to model the document processing workflows, we were forced to create a custom data-structure per multi-tenant process definition and introduce a structural dependency between that data structure and the tasks which decreases reusability of said tasks across process definitions. (2) The Lack of explicit support for multi-tenant customization in BPMN (a) increases the need to add or modify a tenant configuration redundantly at multiple places, limiting the modularity of that configuration, and (b) limits potential future tool support for tenant-specific configuration management. Next, we will elaborate on these findings in detail.

Structural Dependency of Tasks. As discussed in Section 3.3, we have created a composite data structure as a workaround, that is used to pass input document and its meta-data between tasks, because we experienced that BPMN's techniques for passing parameters were not sufficient to realize the requirements set by our document processing application. In order to enable all tenants to read from and write to this data structure, additional structural dependencies between all tasks in our document processing workflow and the common data structure were introduced, i.e. all tasks use (the same) implied knowledge about the common data structure. This workaround is the result of a trade-off in reusability. On the one hand, by introducing this composite data structure and the dependency to the tasks of the multi-tenant process definition, we ensure that the process definition can be efficiently and easily (re-)assembled using existing tasks and graphical tools. On the other hand, in case an additional process definition becomes necessary, tasks of the one process definition cannot be used in the other, as they may rely on different composite data structures for their inter-task communications. For example in the document processing system, tenants with relatively similar requirements are clustered together (tenant

A and tenant B belonging to the same cluster). If however, new tenants show up with very different requirements (thus, belonging to a different cluster), the overall management effort is effectively lower when separating the two clusters in separate process definitions.

Ideally, this problem is addressed at the level of the BPMN language. By supporting the operator to access member variables (in Java, that is the dot operator), the parameter mapping feature, which is configurable using jBPM's graphical tools, *could* be used to pass parameters betweens tasks without the need of additional data structures. Hence, the dependency between tasks in a process definition could be easily managed using graphical tools (to configure the parameter mapping feature) rather than changing the program code of task to comply to the additional data structures.

Explicit Support in BPMN for Tenant-specific Variations. We have observed that BPMN does not explicitly support tenant-specific variations. For the implementation of the document processing application, we therefore borrowed other features of that language to realize the desired level of variability, namely gateways with Java as switch code.

This workaround has two drawbacks: (1) the knowledge about tenant-specific variations for each activity in the document processing workflow is defined within the tenant-specific configuration. As with our implementation, the same knowledge is used when creating the multi-tenant process definition which is a manual process. Thus, our current workaround limits modularity and requires an error-prone manual process. (2) We use the BPMN language item *gateway* to express tenant-specific rather than business-process-driven variability for which it is meant to be used. Therefore, these two semantics become harder to distinguish. As a result, potential tool support for tenant-specific management may be limited.

Note, however, that the lack of explicit support for tenant-specific variations does not affect the manageability of workflow definitions. Placing all tenant-specific variants into a single multi-tenant process definition, i.e. using branches, increases its overall size, and may seem as a bottleneck for (change-)management at first. But, as BPMN supports the partition of workflows into sub-workflows, the size of process definition has no big impact on its practicality in management per-se.

In order to tackle the aforementioned two issues, we envision an extension to BPMN that provides explicit support for tenant-specific variations by introducing two elements. One, an activity that is subject to tenant-specific alternatives should be modelled as variation point[1]. Two, the workflow engine should provide mechanisms to import knowledge about variation points, such as a feature model, and variants from an external source. In our document processing system, this would be the configuration repository. Similar suggestions have been made for the BPEL language but have not been shown in a proof-of-concept implementation, yet [7].

[1] We use the terms *Variation Point* and *Variant* as it is defined in [6].

4.2 Resilience of jBPM's Workflow Execution

The workflow engine jBPM, which we used for our implementation, executes workflows employing dedicated control over the process instance. That is, at the time an execution is triggered, the process definition is loaded into memory. Thereafter, programmatical access to a specific process instance from outside the process instance is very limited, and especially an update of the process definition is not possible.

In addition, for our case, no state during the entire workflow execution is persisted[2]. Technical failures are not considered in the modelling concepts of BPMN. Although jBPM offers technical exception handling[3], it is intended to only run additional procedures in case of exceptions and has no effect on the execution sequence.

Potential Types of Failures. The described properties above lead to following three potential failures of the workflow execution which we will discuss subsequently: (1) A process instance may continue processing on the basis of an out-dated process definition that may lead to task failures or, even worse, to incorrect results of the process. (2) In case a task fails, the entire workflow has to be executed all over. (3) In case the workflow engine crashes, the entire workflow has to be re-run.

The first type of failure can occur when workflow tasks change their scope of activity and, as a result, also the sequence in which the workflow requires to be executed. Example: Assume that a task that was supposed to create and send an e-mail is split into two tasks, one for creating an e-mail, i.e. HTML formatting, BASE64 encoding, etc., and the other for sending the e-mail (talking SMTP with a server). Obviously, process definitions that included this task need to be updated accordingly. Without the ability to update process instances during their execution, all process instances that include that task but have not executed it yet will fail or produce incorrect results.

The second and the third behaviour basically refer to the same issue: In case a task fails, the enclosing workflow is restarted from the beginning. As a result, documents are reprocessed not because of a business process reason, e.g. the document at hand is an exceptional case or contains errors, but purely because of a technical reason. Because we modelled the workflow to process an

[2] The jBPM workflow engine persists workflow state only at so called safe-points. These are phases in which the workflow engine has no further immediate tasks to execute and is waiting for workflow events to continue. For non-interactive workflows that contains only a sequence of subsequent activities, such as the document processing application, no persistence of workflow state is applied during the entire execution.

[3] jBPM distinguishes between two kinds of exceptions: logic and technical exceptions. While logical exceptions refer to exceptional cases in the business logic, e.g. when escalation to the next business hierarchy level is required, technical exceptions can be mapped the exception handling concept found in Java and other programming languages.

entire document processing job (multiple documents) at once, the overhead of a service failure is even higher as already successfully processed documents would be processed again.

This shortcoming is related to jBPM's focus of failure-recovery. It is best suited for situations in which the workflow-engine (or the underlying infrastructure) fails especially when waiting for a particular event to resume the according process instance. This can be a long period when interactive tasks are involved. These phases in which the workflow engine is waiting are called *safe-points*. For our non-interactive and not event-driven workflow, the safe-points are located before the workflow execution has started and after its completion. Thus, our implementation using jBPM makes failures of single tasks expensive[4], as the entire workflow needs to be repeated.

Task Failures in the Context of Distributed SaaS Applications. In the presence of failures with expensive consequences, attention should be paid to the fact that a distributed multi-tenant SaaS application risks multiple natural error sources that may lead activities to fail: First, every distributed system inherently lacks control over the remote machine's state and suffers from occasional data omissions due to network failures. Second, the fact that the benefit from economies-of-scale is a dominant motivation to operate an application on a cloud platform implies that the application is intended to be operated under continuous load. In our case, load refers to document processing jobs that have a SLA-committed completion dates. Thus, large delays in workflow execution are not tolerable from a business perspective.

Conclusion. Therefore, we identify the gap in our document processing SaaS application that it lacks of support for inexpensive failures of tasks. In future work, we plan to elaborate further on safe points that occur between each workflow-step (activity) rendering a process definition to be executed as a set of tasks. Moreover, by removing the centralized control that spans the entire workflow execution and supporting the execution of individual tasks of a workflow from independent workflow engine instances, concurrent execution of semantically parallelizable tasks within a workflow could be enabled. Furthermore, in case a task execution fails, the aforementioned safe points can depict process instance states to resume at when restarting the process instance. In addition, updating the process definition of a running process instance would become simpler, as after each activity the process instance would be in a (persisted) quiescence [8] state and before each activity the process definition is re-read.

Building up on these features, task failures would be less expensive (resume instead of start over) and could therefore be accepted as a planned behaviour of the system and incorporated into the SLA-targeting scheduling strategies. As a result, changes to and failures of the system would be less harmful, and

[4] *Cost* can have multiple dimensions: operational costs, duration (endangering SLA fulfillment) or damage of brand (sending invoices twice and thereby communicating technical error to customers)

the scalability in performance and management overhead (i.e. for re-allocating performance schedules) would benefit significantly.

5 Related Work

Manageability for Business Processes. Modularity is a key concept to support manageability through reuse. Research has been executed to increase the modularity in business process definitions. Geebelen et. al [9] proposed a pre-processing layer for the BPEL standard, that uses a set of concrete parameters to transform a parameterized process definition template into an actual BPEL process definition that can be executed on ordinary BPEL engines. Charfi et. al [10] use aspect orientation to modularize the definition of activities within and across business processes, e.g. an activity that always has to precede another activity can be defined in modularized way. Isotan et. al [11] propose to add composition operators to BPMN in order to facilitate the composition of smaller and reusable definition units into full process definitions. In their work, they are formally modelling operators based on Petri Nets.

In contrast, our document processing application is designed to be operated as SaaS application and, thereby, has a different set of requirements for manageability: We address one application domain at a time by creating a single process definition including all anticipated variabilities. Not dealing with a large amount of separate process definitions, our context benefits less from the kind of modularity that is presented in the related work. We rather lack of reusability of tasks across process definitions, as elaborated in Section 4.

Multi-Tenancy for Business Processes. Pathirage et. al [12] address multi-tenancy mostly at the infrastructure level. They provide a platform on top of which a BPEL engine can be run and that maintains a tenant context during the entire workflow execution.

However, it does not take customization of workflows into account, i.e. all tenants operate on the same workflow, while we focus on workflow customization as well as multi-tenancy.

Variability for Business Processes. The work of Mietzner et. al [13] focus mainly on modelling variability and providing deployment support in that it will choose the set of required components optimizing for the lowest operational costs.

While they present rather generic concepts for modelling workflow variability, our work is based on a practical experience with a concrete state-of-practice framework from which we extract further challenges. We also focus on the business process modelling aspect that is not enlightened in their work.

Geebelen et. al present in a later work [14] a framework for run-time adaptation of workflow processes. They use an application-domain-dependent workflow template at which concrete service calls are weaved in at run-time depending on an external policy engine. Also rollbacks to previous workflow activities are provided.

Even though, they provide manageable flexibility, their scope of workflow customization differs from ours. While they provide a fixed sequence of activities and flexibility in choosing the service to execute an activity, we offer alternative sequences based on tenant-specific requirements.

VxBPEL [7] is an extension to BPEL that explicitly adds alternative task implementations to an activity in the process definition. It is motivated that the knowledge about variations (1) should be obtained from an external source and (2) should be injectable into on-going workflow executions. The implementation of that work does not show those motivated proposals.

In their proposals, the authors argue similarly to us. Yet, we differ in the fact that we use business process modelling for non-interactive batch-processing and variability in the context of multi-tenancy, while they argue on basis of interactive application and introduce variability to achieve higher Quality-of-Service. Furthermore, we use the state-of-practice technology jBPM without modifications in order to comply with cloud providers as well as with existing applications and tools.

Resilience of Workflow Execution. In Section 4.2, we described the kind of resilience for workflow executions that is required for the document processing application, i.e. a task execution failure should not cause the enclosing workflow to be reset to the beginning.

Leymann et. al [15] describe a workflow management system that is based on multiple message queues. They claim that their system, persisting state information about each invoked task, is "forward recoverable". While their system is situated in a local environment with (remote) interactive clients, our context is a non-interactive workflow as distributed SaaS application.

The work of Yu et. al [16] proposes to process BPEL workflows without a central execution engine. One of its key goals is to enable dynamically composed workflows which also addresses changes in the task execution sequence in the presence of service failures. They make use of continuations which are persisted after each task execution of the workflow and that can be picked-up by different workflow engines for continuing execution, and extend the BPEL execution engine. We, the other hand, use state-of-practice tools to model and execute workflows.

6 Conclusion

We presented our experiences with the implementation of a customizable multi-tenant SaaS application for document processing. We discussed the key requirements for this application: multi-tenancy and Software-as-a-Service on the one hand, and customizability to tenant-specific requirements on the other hand. To guarantee the practical relevance of our findings, we addressed these requirements in the context of state-of-practice technologies from the JBoss family. Specifically, we employed Business Process Modelling and Notation (BPMN) language to model tenant-specific customizable business processes of the document processing system and jBPM for their execution.

Our findings can be summarized as follows. First, because of the parameter-passing mechanism that is currently provided in BPMN, it is hard to design individual tasks in a modular manner so that they can be reused across business process definitions. Our second finding is a consequence of the fact that BPMN lacks explicit support for multi-tenancy and variability. We introduce workflow branches to express tenant-specific instead of business-case variabilities as a workaround. Ideally, the workflow modelling language should offer support to describe these tenant-level variabilities explicitly. In our third finding, we have argued that using a centralized run-time instance to control the entire workflow may not provide the necessary resilience in execution, because of the high costs related to recover from task failure. It may therefore only be partially suited for SaaS environments where different kinds of faults are likely to occur regularly. Their occurrence may endanger SLA commitments and thereby limit scalability of the application.

Customization will gain importance in multi-tenant Software-as-a-Service applications as it enables the SaaS provider to fine-tune his offerings to specific tenants without losing the benefits of scale inherent to SaaS. Not only applications and services, but also the composition of those, i.e. a workflow-driven application, need to support customization to facilitate the migration of legacy applications to the cloud.

Acknowledgments This research is partially funded by the Research Fund KU Leuven and by the iMinds project CUSTOMSS. The iMinds CUSTOMSS is a project co-funded by iMinds (Interdisciplinary institute for Technology) a research institute founded by the Flemish Government. Companies and organizations involved in the project are AGFA Healthcare, IBCN/INTEC-UGent, Televic Healthcare, and UnifiedPost.

References

1. iMinds CUSTOMSS Project Consortium: iMinds CUSTOMSS Project (2013), `http://distrinet.cs.kuleuven.be/research/projects/showProject.do?projectID=CUSTOMSS`
2. The jBPM Team of the JBoss Community: jBPM (2013), `http://www.jboss.org/jbpm`
3. The JBoss Community: JBoss AS7 (2013), `http://www.jboss.org/as7`
4. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document (1990)
5. Walraven, S., Truyen, E., Joosen, W.: A Middleware Layer for Flexible and Cost-Efficient Multi-tenant Applications. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 370–389. Springer, Heidelberg (2011)
6. Chang, S.H., Kim, S.D.: A Variability Modeling Method for Adaptable Services in Service-Oriented Computing. In: 11th International Software Product Line Conference, 2007. SPLC 2007, pp. 261–268 (2007)
7. Koning, M., ai˜Sun, C., Sinnema, M., Avgeriou, P.: Vxbpel: Supporting variability for web services in bpel. Information and Software Technology 51, 258–269 (2009)

8. Kramer, J., Magee, J.: The evolving philosophers problem: dynamic change management. IEEE Transactions on Software Engineering 16, 1293–1306 (1990)
9. Geebelen, K., Michiels, S., Joosen, W.: Dynamic reconfiguration using template based web service composition. In: Proceedings of the 3rd workshop on Middleware for service oriented computing. MW4SOC '08, pp. 49–54. ACM Press, New York (2008)
10. Charfi, A., Awasthi, P.: Aspect-oriented web service composition with AO4BPEL. In (LJ) Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 168–182. Springer, Heidelberg (2004)
11. Istoan, P.: Defining composition operators for bpmn. In: Gschwind, T., De Paoli, F., Gruhn, V., Book, M. (eds.) SC 2012. LNCS, vol. 7306, pp. 17–34. Springer, Heidelberg (2012)
12. Pathirage, M., Perera, S., Kumara, I., Weerawarana, S.: A multi-tenant architecture for business process executions. In: IEEE International Conference on Web Services (ICWS), pp. 121–128 (2011)
13. Mietzner, R., Metzger, A., Leymann, F., Pohl, K.: Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems. PESOS '09, Washington, DC, USA, pp. 18–25. IEEE Computer Society Press, Los Alamitos (2009)
14. Geebelen, K., Kulikowski, E., Truyen, E., Joosen, W.: A mvc framework for policy-based adaptation of workflow processes: A case study on confidentiality. In: 2010 IEEE International Conference on Web Services (ICWS), pp. 401–408 (2010)
15. Leymnn, F., Roller, D.: Building a robust workflow management system with persistent queues and stored procedures. In: 14th International Conference on Data Engineering, Proceedings, pp. 254–258 (1998)
16. Yu, W.: Running BPEL Processes without Central Engines 1, 224 (2007)