α BPaaS - A Customizable BPaaS on the Cloud*

Yehia Taher1,3, Rafiqul Haque2, Willem-Jan van den Heuvel3 and Beatrice Finance1

1Laboratoire PRiSM, Université Versailles ST Quentin-En-Yvelines, Versailles, France
2Lero - The Irish Software Engineering Research Centre, University of Limerick, Limerick, Ireland
3ERISS, Tilburg University, Tilburg, The Netherlands

Keywords: Business Process as a Service, Cloud Computing, Customization.

Abstract: With the emergence of the Internet technologies - notably, SOA and Cloud Computing - enterprises are increasingly delivering their business services through on-line service offerings called Business Process as a Service (BPaaS). However, current BPaaS offerings can be perceived as monolithic cloud solutions that are constrained by the capabilities that are made available by the provider at their delivery level and do not allow for easy extensibility or customization options. In this paper, we propose a novel BPaaS engineering techniques which cater for the tailoring of services to specific business needs using a mixture of SaaS, PaaS and IaaS solutions - possibly from various providers. It is the prime goal of our proposal to simplify the engineering of BPaaS applications by hiding the complexity of their development and deployment. This is achieved by providing a customization solution to manage all configuration of functional and non functional aspects related to a BPaaS offering.

1 INTRODUCTION

Recently, Business Process as a Service (BPaaS) has drawn enormous attention from the software vendors, IT professional, and researchers alike. Gartner is predicting that Business Process as a Service will grow from $ 84.1B in 2012 to $ 144B in 2016, generating a global compound annual growth rate of 15% (Columbus, 2012). Forecasts from industry pundits such as Gartner about BPaaS have catalyzed investments of software vendors in developing the innovative BPaaS solutions. Indeed, today virtually all software vendors such as IBM market their own BPaaS offerings.

Business Process as a Service may be informally defined as any type of horizontal (generic) or vertical (domain-specific) business process that is delivered on the cloud services model (J. Hurwitz and Kirch, 2012). Currently, a BPaaS is delivered as a comprehensive integrated suite comprising not only Business Process services but also the Software Services (SaaSs), Platform Services (PaaSs), and Infrastructure Services (IaaSs) that enact them. In this way, the BPaaS suites aim at provisioning a highly standardized, comprehensive and cost-effective solution.

The central notion of a BPaaS suite is a business process. A business process is composed of a set of activities that serve the specific purposes of an organization (Leymann and Roller, 2000). The activities perform operations that must produce outcomes desired by an organization. Since a business process is specific to a enterprise context, it should thus meet the local functional requirements of an organization. Furthermore, a business process should be operated under the constraint of Quality of Services (QoSs) conditions and business policies. The QoSs and business policies promote the context specificity of the business processes because the quality parameters and their values, and the policies heavily depend on the preference of an organization or individual.

McCue (McCue, 2012) emphasizes the critical need for customizing the business processes in a BPaaS suite. Enterprise-specific requirements cannot be met unless the standard business process services in BPaaS solutions are tuned accordingly. Customization is not restricted to business process services only, but in fact propagates to the underlying SaaS, PaaS and IaaS services. For example, adding an activity in the business process may incur the need of a novel software service to realize that activity and new IaaS services to ascertain persistency.

The above considerations highlight the need for customization facilities. Currently, the BPaaS suites do not offer customization service. The BPaaS suite providers may offer the customization as an additional service to their clients, however, it is not cost effective from time and economic perspective. More-

---

*This research has been partially funded by the 4CaaSt project.
over, offshore customization is difficult to manage; and how a business process can be customized by adapting the internal business policies that are confidential for an organization - is a major challenge for the offshore customization approach. The on-premises customization approach is more efficient than the offshore customization in terms of security, manageability and cost. Therefore, an on-premises customization service is a strong requirement but missing in the state of the art. This research is aimed to address this requirement.

The goal of the research presented in this paper is to propose an advanced Business Process as a Service (αBPaaS) suite containing a customization service that will enable tailoring the functional, non-functional aspects of business processes to meet the enterprise-specific requirements. The customization service will also assist in systematically tuning the software-, platform-, and infrastructure services that realize a particular business process. The proposed customization component is integrated with the αBPaaS suite which is more customer-faced or user-centric than the existing BPaaS suites.

The remainder of this paper is organized as follows. A brief overview of αBPaaS is presented in Section 2. Then, the customization mechanisms are described in Section 3. Section 4 explores and partially validates the customization mechanisms with an usage scenario. Section 5 presents the conclusion and future works.

2 αBPaaS - IN A NUTSHELL

Traditionally a cloud service provider offers a BPaaS suite that incorporates business processes, the software services that contain methods for performing operations, the platform services, and the physical resources. These standard services are packaged as a monolithic service stack as shown in the figure 1. However a service client may need to customize it for satisfying some specific needs. Today, the customization service is absent, a BPaaS suite is not a customization-capable solution.

To address this problem, the αBPaaS, a customization-capable solution, is proposed. “α” stands for “advanced” and indicates the advancement of the BPaaS suites which is aimed in this paper. The αBPaaS suite is depicted in the figure 2. Its role is to assist the users in customizing the quality parameters, policies, and functional properties of the services in the bundle. The key idea behind is to cloudify the notion of customization and to deliver it as a service.

Different approaches has been proposed for producing outcomes through customization (Zhu and Zheng, 2005), (Stollberg and Muth, 2009). A Multi-Layered Approach (Taher et al., 2011) - which produces multi-level solutions - is adopted in αBPaaS. In such an approach, at each layer, the customization produces a solution. αBPaaS offers a set of operators that assist in performing multi-level customization for localizing or personalizing the services or processes. In this way, it caters for customizing services or processes by fine-tuning the policies and quality parameters. The customization of policies and parameters produces an end-solution that fits to specific functional areas of a context. This aspect will be demonstrated in Section 3.

2.1 αBPaaS Architecture

The αBPaaS solution architecture is introduced in figure 3. Typically, a BPaaS suite is a bundle of BPaaS, SaaS, PaaS, and IaaS where BPaaS resides on the top of all other services. In our architecture, these services are decoupled and offered as different packages through APIs. This separation of services facilitates the customization of services efficiently and correctly. A customization front-end is integrated in the architecture. This front-end provides the customization in-
3 CUSTOMIZATION COMPONENT

The customization component offers a list of operators to assist the users in performing their customization tasks. It underpins the specification of the data properties and the behavioural properties of the services. Its formal definition is defined in the following. First an intuitive definition of αBPaaS is given:

Definition 1. The αBPaaS can be defined as αBPaaS = (NFp, Fp, Bp, Sp, Rp, λ, C) such that,

- NFp, denotes the non-functional aspects of the services or processes; NFp = (Ψ, ρ) where, Ψ and ρ represents the policies and the quality parameters of the relevant services or process activities respectively.
- Fp denotes the functional aspects in particular, the functional properties of services or processes.
- Bp is the business process. The business process is defined as Bp = (A, P, P, E) where,
  - A is the business process activity.
  - P represents the control flow patterns such as Parallel, Sequence.
  - P represents the interaction pattern such as synchronous, asynchronous interaction patterns.
  - E is the events such as a messaging event.
- Sp represents software services (also called software components).
- Rp is the physical resource services such as virtual machine, storage.
- λ is the platform service.
- C is the customization instructions.

3.1 Data Property Specification

The αBPaaS suite introduces a hard/soft and a vital/non-vital types that are very different from the classical data types. They allow the definition of constraints. They are briefly described here.

- Hard: Hard denotes the strict constraint that cannot be violated under any circumstance. For instance, authentication checking is a security policy that can be characterised as a strict constraint by specifying its ‘type’ ‘hard’. Consequently, at runtime, each instance accessing a data storage must pass the authentication check.
- Soft: Soft denotes a weaker constraint. Its violation does not bring severe consequence for a running instance. For instance, a violation of a soft constraint does not influence the complete failure of a running instance.

These two constraints influence the policies and quality parameters. Their constraints definition is given below. Note that FC and FP represent respectively the complete and partial failure, and ‘V’ denotes violation.

- ∀(ψ ∈ Ψ), IF (ψ.Type = “Hard”) THEN V(ψ) → FC ELSE IF (ψ.Type = “Soft”) THEN V(ψ) → FP.
- ∀(ρ ∈ ρ), IF (ρ.Type = “Hard”) THEN V(ρ) → FC ELSE IF (ρ.Type = “Soft”) THEN V(ρ) → FP.

- Vital: Vital denotes the strict constraint of a business process activity. It is used for characterising the activities in a business process. If an activity is ‘vital’ then the failure of that activity affects all other activities contained in the business process. As an example, if process payment is of a vital type and if this activity fails, then all completed activities such as process delivery will be forced to fail.
- Non-Vital: Non-Vital denotes the weaker constraint of a business process activity. The failure of a ‘non-vital’ activity may not affect other activities contained in the business process. For instance, in a travel management business process, if hiring car is of non-vital type then the failure of this activity will not affect other activities such as reserve hotel room and book flight.
The predicates below show a vital and non-vital type activity with their respective complete or partial failure.

\(- \forall (A_i \in B^p), \ IF (A_i.Type = "Vital") \ THEN \ F(A_i) \rightarrow ignored(V).\)

\(- \forall (\psi_i \in \Psi), \ IF (\psi_i.Type = "Soft") \ THEN \ V(\psi_i) \rightarrow ignored(V).\)

- **retry**: Retry forces the system to retry the failed operation upon failure of that operation. The operation is retried until it is successfully completed or it reaches the maximum-limit for retrying. The logical condition is defined below:

\(- \forall (O_i \in A_i), \ IF (O_i \rightarrow Failed) \ THEN \ retryN(O_i), \ where \ ‘N’ \ denotes \ the \ retry \ limit.\)

- **ignore**: Ignore forces the system to ignore the failure of an activity in the business process. In addition, this primitive allows the system to ignore the violation of soft constraints. The logical conditions for ignoring the failure of activity and soft constraint violation are given below:

\(- \forall (A_i \in B^p), \ IF (A_i.Type = "Vital") \ THEN \ F(A_i) \rightarrow ignored(V).\)

- **wait**: Wait is used for specifying the waiting time between the business process activities. The primitive forces a business process activity to wait for another activity to be completed until the waiting time elapsed.

\(- \forall (O_i \in A_i), \ IF (O_i \rightarrow Failed) \ THEN \ retryN(O_i), \ where \ ‘N’ \ denotes \ the \ retry \ limit.\)

\(- \forall (\psi_i \in \Psi), \ IF (\psi_i.Type = "Soft") \ THEN \ V(\psi_i) \rightarrow ignored(V).\)

- **repair**: Repair prevents the complete failure of a business process. If an activity is vital and if an operation is performed by that activity has failed then, the repair operation is performed and healed the failed activity. The logical condition for repairing a failed activity is stated below:

\(- \forall (A_i \in B^p), \ IF (A_i.Type = "Vital") \ THEN \ repair (O_i) \rightarrow repaired(O_i), \ where \ O_i \ denotes \ an \ operation.\)

In principle, it is not always true that the failed operation will be successfully repaired always however, in this condition, we have considered what should happen in all ideal cases. Note that, Repair can be a composite operation in some cases. For instance, if a software service is permanently unavailable, the repair operation is performed by combining the substitute and invoke operations: (substitute ρ invoke) \ \implies \ repair. The substitute operation is defined later in this section. Invoke denotes activating a process activity. The invocation occurs after finishing the substitute operation.

- **remove**: In this condition, we have considered what should happen in all ideal cases. Note that, Remove can be a composite operation in some cases. For instance, if a software service is permanently unavailable, the repair operation is performed by combining the substitute and invoke operations: (substitute ρ invoke) \ \implies \ repair. The substitute operation is defined later in this section. Invoke denotes activating a process activity. The invocation occurs after finishing the substitute operation.

- **add**: In this condition, we have considered what should happen in all ideal cases. Note that, Add can be a composite operation in some cases. For instance, if a software service is permanently unavailable, the repair operation is performed by combining the substitute and invoke operations: (substitute ρ invoke) \ \implies \ repair. The substitute operation is defined later in this section. Invoke denotes activating a process activity. The invocation occurs after finishing the substitute operation.

- **decompose**: In this condition, we have considered what should happen in all ideal cases. Note that, Decompose can be a composite operation in some cases. For instance, if a software service is permanently unavailable, the repair operation is performed by combining the substitute and invoke operations: (substitute ρ invoke) \ \implies \ repair. The substitute operation is defined later in this section. Invoke denotes activating a process activity. The invocation occurs after finishing the substitute operation.

3.3 Operators

A list of operators are offered for assisting the users in customizing services and in producing different levels of solutions. These operators are the basic building blocks of the customization component. They are described below.

- **decompose**: The parameters, business process activities can be decomposed into ‘n’ (must be greater than 1) number of parameters, process activities using decompose operator. The following show the decomposition operations.

\(- \ decompose (p) = p_1 \land p_2 \ldots \land p_n. \ Decomposition \ of \ a \ parameter \ produces \ ‘n’ \ number \ of \ parameters.\)

\(- \ decompose (A) = a_1 \land a_2 \ldots \land a_n. \ Decomposition \ of \ an \ activity \ generates \ ‘n’ \ number \ of \ activities.\)

It is worth noting that, the value of ‘n’ means, the number of decomposed elements in particular, the parameters or activities essentially depends on the requirements of the users.

- **add** and **remove**: The add and remove operators used for performing respectively addition and removal operations on process activities and events, resources, add-ons, quality parameters, etc. These operations performed on BPaas suite are not limited to the ones shown below.

\(- \ add \ or \ remove (A \lor P^c \lor P^l \lor E \lor \psi \lor p) \rightarrow B^c_p. \ In \ this \ operation, \ a \ business \ process \ is \ customized \ by \ adding \ or \ deleting \ an \ element, \ quality \ parameters, \ or \ policies. \ B^c_p \ denotes \ the \ new \ version \ of \ the \ business \ process.\)

\(- \ add \ or \ remove (M \lor param \lor (O \land param) \lor \psi \lor p) \rightarrow S^c_p. \ ‘M’ \ denotes \ method \ that \ realizes \ the \ process \ activities. \ A \ software \ service \ is \ customized \ by \ adding \ or \ deleting \ operations \ (methods) \ or \ parameters \ or \ both. \ For \ software \ services, \ a \ user \ may \ customize \ the \ quality \ parameters \ or \ policies \ in \ particular, \ the \ security \ policy \ ‘authentication’. \ S^c_p \ denotes \ the \ new \ version \ of \ the \ software \ service \ produced \ after \ performing \ the \ add \ operation.\)

\(- \ add \ or \ remove (V^M \lor M^m \lor S^x \lor Nw) \rightarrow R^p. \ Where \ V^M, \ M^m, \ S^x, \ and \ Nw \ represent \ virtual \ activities.\)
machine, memory, storage, and virtual local area network respectively. $R'_{\alpha}$ denotes the new version of the resource services.

- **add or remove** $(W^S \lor D \lor (Plgn \lor L^S)) \rightarrow \lambda_n$. Where $W^S$ and $D$ represent Web Server and Database and $Plgn$ and $L^S$ denotes plug-in and class library that are added or deleted in the development tool of the platform services. $\lambda_n$ denotes the new version of the platform service.

- **substitute**: Substitute operator used to replace the process activities, software services, platform services, and resource services. The substitute operation is a composite operation: $(\text{remove} \circ \text{find} \circ \text{add}) \rightarrow \text{substitute}$. Add and remove operations have already been defined and the Find operation denotes performing search on a service repository to find a service which is functionally equivalent to the service that is to be replaced. The following examples show the substitute operations:
  - **substitute** $(A \lor E) \rightarrow B_{\alpha}^P$. The substitute operation replaces an activity or event in a business process.
  - **substitute** $(s) \rightarrow S'_{\alpha}$. Here, the substitute operation replaces a service by a new service.
  - **substitute** $((Plgn \lor L^S) \lor W^S \lor D) \rightarrow \lambda_n$. An element can be substitute in the platform services using this operator.
  - **substitute** $(V^M \lor M^m \lor S^t \lor Nw) \rightarrow R'_{\alpha}$. A virtual machine or storage or other service elements can be replaced by the substitute operation.

- **Create**: Create is a special operator used for performing operations on data storage service. Create operator is used to create instances such as buckets(containers) in a storage service. The create operation is shown below.
  - **Create(bucket \lor object) \rightarrow \lambda_n**. Note that, storage service is a type of PaaS.

- **rename**: Rename is a simple operation that is performed for modifying the name of a business process activity or event.
  - **rename(A \lor E) \rightarrow B_{\alpha}^P**.

- **refine**: The refinement operations is performed on a BPaaS suite to fine-tune the BPaaS suite. The refine operator is used for performing refinement operation. It is used for adding and removing business process activities, parameters, events etc. Software services, resource services can be refined using this operator.

Furthermore, the $\alpha$BPaaS customization component contains instructions that assist in selecting the operators for customizing the BPaaS suite. The customization wizard of the $\alpha$BPaaS suite provides the step-by-step instructions for customizing the services. They are presented on the screen based on the context specific or function specific inputs keyed by the users.

## 4 USE CASE SCENARIO

This section illustrates a concrete scenario how the $\alpha$BPaaS suite supports the user in customizing of a cloud based BPaaS suite. Figure 4 depicts the use of $\alpha$BPaaS on a scenario and shows the different solutions produced at the different levels.

The customization starts with customizing the business process service. The meta solution is the most generic business process offered in the BPaaS suite. The generic business process contains the basic activities, events and control flows such as `create and send PO (PO stands for Purchase Order), Process PO, etc.`

The generic business process is customized according to the requirements of a domain in order to produce the domain specific solution. In our scenario, we have picked automotive business domain. The customization begins with decomposing `create and send PO into create PO and send PO activities. The approve PO is a new activity added between these activities. The control flow between these activities is customized as well. A messaging event Receive PO is added in the domain specific process and the activity Process PO is decomposed into check inventory and Register PO. Considering the check inventory activity, only the ideal case is considered in this example. The decomposition of this activity can produce more activities such as `replenish product`.

A company “ABS Inc.” is considered the context in this example. The domain specific solution needs to be customized to produced the context specific solution of the company. At this phase, two swimlanes are introduced to separate the business processes that represent buying and selling business processes. The message flows (dotted arrow lines) are added in the process; the buyer and seller interacts through the messages. The messaging events `send approval notification` and `send invoice` are added in the seller’s process. The control flows between these events are added. The activity `Deliver PO` is renamed `Process Delivery`. On the other hand, `Receive PO approval notification`, `Receive Delivery`, and `Receive Invoice` messaging events are added in buying process. The flows are added for connecting these events. This customization produces a business process that meets the requirements of the ABS Inc.
Figure 4: A Usage Scenario for the αBPaaS.
In order to make sure the business process meets the requirements of a specific functional area or module, yet another customization is performed. The payment area is extended by adding Calculate Price activity. Moreover, the process delivery activity is decomposed into send delivery information and receive shipment detail. A swimlane is added for capturing the shipment activities. This swimlane represents a shipment process that is executed and managed by a shipper. A list of events and activities which include receive delivery information, process shipment, ship order, send shipment detail are added in the shipment process.

Furthermore, at this customization phase, the properties associating with the business process activities are specified. The activity type Vital, NonVital and the policies and quality parameters related to the business process activities are defined. Approval notification time is a policy assigned to send approval notification with a data property “time” associating a value “24”. The policy is characterised by the data property “Soft” that denotes if the notification is not sent within 24 hours; the process execution will not be interrupted. However, the “hard” type policy would stop the execution of the process if the policy was not satisfied. Check Inventory activity is associated by the “hard” type policy. Moreover, the functional properties repair, retry, and wait are specified for different activities in this functional area specific process.

As this is the final phase of customizing BPaaS suite, the software, platform, and infrastructure services are customized in this phase. In the given example, customization of these services are shown inside a block of dotted lines.

For the software components (in SaaS), the quality parameters are defined. Availability and Reliability are the parameters defined for software services along with their values “24” hours and “High”. The type of these parameters is “hard” denotes that these values must be satisfied at runtime. It is worth noting that, the lower level services should be customized when a user confirms that the functional area specific business process contains all the required activities.

Considering the customization of software components, the simplest case is considered for this example. Considering the customization of PaaS, a Google Web Server is added in the platform. Additionally, the SampleDB is substituted by the CouchDB. The BPEL 2.0 plug-in and jcloud library are added in the development tool that is integrated in the platform service as a package. Finally, the infrastructure services are customized. Two Virtual Machine images provided by the Amazon Inc. are added in the infrastructure to increase the computation capability of the system. In addition, a Cloud Storage provided by the Rackspace Cloud is added in the infrastructure to enhance the size of the storage.

After finishing the customization tasks, the business process, the software components, the platform, and the physical resources must be assembled together. The assemble is not shown in the example, however, it is the last phase in the customization life cycle.

5 CONCLUSIONS

This paper introduced a novel approach for customizing BPaaS following the multi-layered approach that allows fine tuning the BPaaS solutions to satisfy the enterprise specific requirements. One of the unique characteristics of our approach is, it underpins tailoring not only the business process services, but also the SaaS, PaaS and IaaS that implement the business process services in a transparent, tractable and structured manner. The research results in this paper are core results in nature. Extensions and improvements are needed to validate the customization life-cycle and mechanisms. The proposed operators will have to be formalized in near future.

REFERENCES

Columbus, L. (2012). Forecasing public cloud adoption in the enterprise.


