Transactional Behavior Verification in Business Process as a Service Configuration

Scott Bourne, Claudia Szabo, *Member, IEEE*, Quan Z. Sheng, *Member, IEEE*

Abstract—Business Process as a Service (BPaaS) is an emerging type of cloud service that offers configurable and executable business processes to clients over the Internet. As BPaaS is still in early years of research, many open issues remain. Managing the configuration of BPaaS builds on areas such as software product lines and configurable business processes. The problem has concerns to consider from several perspectives, such as the different types of variable features, constraints between configuration options, and satisfying the requirements provided by the client. In our approach, we use temporal logic templates to elicit transactional requirements from clients that the configured service must adhere to. For formalizing constraints over configuration, feature models are used. To manage all these concerns during BPaaS configuration, we develop a structured process that applies formal methods while directing clients through specifying transactional requirements and selecting configurable features. The *Binary Decision Diagram* (BDD) analysis is then used to verify that the selected configurable features do not violate any constraints. Finally, model checking is applied to verify the configured service against the transactional requirement set. We demonstrate the feasibility of our approach with several validation scenarios and performance evaluations.

Index Terms—Business Process as a Service, formal methods, verification, transactional requirements, model checking

1 Introduction

In recent years, cloud services have had dramatic impacts in both the research [1] and industry [2] landscapes of service-oriented computing. Cloud computing has become a popular paradigm for delivering a wide range of services, such as software applications, computing capacity, storage, and virtual platforms [3]. Cloud service providers can offer these utilities to clients over the Internet in a pay-by-use manner. The distinctive properties of cloud services include:

- On-demand availability through public or private network access, most commonly the Internet [3].
- Utilization of pooled resources such as servers, applications, CPU time, or storage.
- Dynamic response to workload by elastically provisioning and releasing resources [3][4].
- Configurability of service behavior of properties to meet individual client requirements [5][6][7].

The traditional hierarchy of cloud service types is comprised of three layers, where each layer can provide the *base* (infrastructure or platform) for running services within the layer above [3]. *Infrastructure as a Service* (IaaS) is the bottom service layer, providing access to virtualized physical resources, such as storage and computation capacity. Computing capacity offered by Amazon EC2[1] or IBM SmartCloud Enterprise+[2] are examples of IaaS offerings. *Platform as a Service* (PaaS) provides access to utilities such as software development and hosting frameworks. For example, Google App Engine[3] and Microsoft Azure[4] both contain PaaS features for web application development and hosting. Finally, *Software as a Service* (SaaS) are software applications deployed in a way that is Internet accessible, automatically scaling, and multi-tenant. SaaS enables clients to remotely use software complex systems, such as customer relationship management through Salseforce[5].

A proposed *fourth* layer of the cloud service architecture residing above SaaS has been in the form of *Business Process as a Service* (BPaaS), which has had increasing research interest in recent years [8][9][10][11]. The driving idea behind BPaaS is to mash-up services from numerous providers into a business process structure, which can then be offered to clients as its own service. BPaaS providers will naturally target common or proven business processes that apply to a large potential market, or require management of several complex components. This is appealing to clients as it provides a low-cost, low-risk outsource option for integral business operations.

Figure 1 shows an abstract example that demonstrates the structure and variety of services and resources that a BPaaS can utilize. In this example, the BPaaS is composed of heterogeneous component services from the service provider and third parties. Two SaaS services (i.e., SaaS 1 and SaaS 2) used by the

- *S. Bourne and C. Szabo are with the School of Computer Science, the University of Adelaide, SA 5005, Australia. E-mail: {scott.bourne, claudia.szabo}@adelaide.edu.au*
- *Quan Z. Sheng is with the Department of Computing, Macquarie University, NSW 2109, Australia. E-mail: michael.sheng@mq.edu.au*

1. https://aws.amazon.com/ec2/
2. http://www-07.ibm.com/au/managed-cloud-hosting/
3. https://cloud.google.com/appengine/docs
4. https://azure.microsoft.com/en-gb/
5. www.salesforce.com/au/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

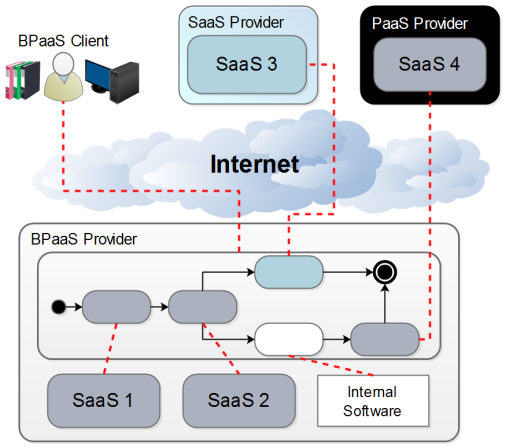


Fig. 1. An abstract BPaaS example

BPaaS are hosted and managed by the same provider. Private internal software exclusive to the BPaaS is also required. Two of the SaaS services are from external sources - SaaS 3 is from a third party, while SaaS 4 is another service of the BPaaS provider, but hosted on an external PaaS.

Configurability is a key property for BPaaS, similar to all services in the cloud hierarchy. The use of configurable business processes can affect the actual properties of the business process, such as the workflow structure [12][13], resources used [14][15], and variables [16]. In contrast, the use of configurable BPaaS, due to their service-based nature, ensures that clients can align BPaaS services with their internal business operations and policies, while providers can exploit economies of scale by offering their service to a larger potential market. Moreover, BPaaS introduces third-party services and with this additional transactional concerns when compared to traditional configurable business process mainly stemming from the fact that transactional requirements will span more than one service provider and will have to adhere to various provider constraints. The inclusion of the client's perspective to the traditional configurable business process paradigm by allowing the client to configure more than just selected features or control flow variations makes BPaaS services appealing to a variety of client types [8][9][10][11], while at the same time introducing challenges as discussed below. For example, some business clients may have internal systems already in place to handle certain steps, while smaller businesses may outsource these services.

However, when a BPaaS has a large number of configurable components, the verification that the behavior is correct and/or meets client requirements can be challenging, as state space explosion hinders the verification of large models [17][18], and the requirements provided by clients can be complex [19]. Existing approaches in managing business process configuration ensure *domain constraints* over configuration choices, while allowing basic client requirements such as selected features or control flow variations. One area that has yet to receive research attention is ensuring both *domain constraints* and client *transactional requirements* during BPaaS configuration. These requirements can include conditions for acceptable process commit or abortion, required recovery operations for key activities, or valid forms of process compensation, and are difficult to verify in a cloud-based scenario where multiple stakeholders are involved. A configuration method that ensures complex requirements within a feasible runtime will be able to provide service clients with increased trust for outsourcing potentially sensitive business operations.

To address these problems, we propose a three-step configuration and verification process which relies on a modeling paradigm. Such paradigm allows us to capture transactional requirements and subsequently verify them. Our approach is expressive and relatively easy to use by stakeholders, while at the same time being sufficiently rigorous to allow us to apply formal methods for verification.

The remainder of this paper is organized as follows. Section 2 provides the details of our configurable BPaaS model, with formalization of domain constraints and client transactional requirements. A configuration process outlined in Section 3 applies formal methods to these models to determine a configuration solution. Section 4 contains a report of our implementation, validation scenarios, and performance analysis with large models. Section 5 discusses future directions for our work. Finally, Section 6 overviews the related research and Section 7 provides some concluding remarks.

2 Transactional BPaaS Modeling

Before transactional requirements can be verified, they need to be modeled within the BPaaS context. To better illustrate our approach, we consider the scenario of a configurable Web store checkout BPaaS. The clients of this process will be small and medium sized Web stores, while the users will be customers. This BPaaS targets businesses selling physical or digital goods, as standard orders or pre-orders. The process places shipping orders for physical goods and retrieves download links for digital goods. Specific tasks in the process include validating customers, obtaining payment details, updating inventory and accounting systems, and processing customer payment amongst others. Clients can configure the structure of this process to suit their business requirements. For example, stores only selling digital goods can remove all tasks related to product shipping, while stores that do not store customer details can restrict the process to handling unregistered guest customers.

This process provisions both constant and configurable external resources. Examples of configurable resources include the optional payment services used

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

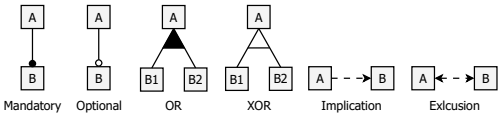


Fig. 2. Feature model constraints used in our approach

by the BPaaS, such as Paypal[6], or Epoch[7] and eWay[8] credit card transactional managers. Data objects can also be configured as appropriate, such as including or excluding fields in customer and product details.

2.1 Modeling Domain Constraints

Domain constraints are rules that allow providers to restrict BPaaS configuration to valid choices. For example, several credit card transaction management resources may be available for a given payment operation, but at least one must be selected in any configuration. Our BPaaS modeling approach adapts *feature models* from the software product line engineering domain [20]. Feature models are tree-like structures being able to express domain constraints formally and visually. Typically, feature models are used to express variability in a configurable system, by modeling the constraints between optional *features*. In our approach, we adapt them to formalize constraints between the configurable activities, resources, and data objects of a BPaaS. Furthermore, by using one feature model to define all BPaaS domain constraints, we are able to define constraints that cross these configuration perspectives. For example, certain configurable activities may require the selection of specific resources.

We apply six feature model relationship structures, shown in Figure 2, to model domain constraints. The first four relationships apply to one or more leaf features if the head feature is selected. For example, the *Mandatory* and *Optional* structures define that if feature `A` is selected, then feature `B` is essential or optional respectively. *Implication* and *Exclusion* can be defined between any two features in the model, regardless of their level in the tree structure.

A feature model capturing the domain constraints for the configuration of the checkout BPaaS is shown in Figure 3. The root `Checkout Service` feature contains all other features as children, and allows constraints to cross between activity, resource, and data object perspectives. For example, `Validate Login` is an optional feature, but it requires `Register User`, `Retrieve User Details`, either `Private Customer Repository` or `Provider Storage`, and enables `Store Payment Details` to be selected. A selection of features that satisfy all constraints in this model, therefore conforming to all configuration requirements of the provider, is a *valid configuration* of the BPaaS.

6. https://developer.paypal.com/
7. https://www.epoch.com/en/index.html
8. http://www.eway.com.au/developers/api/overview

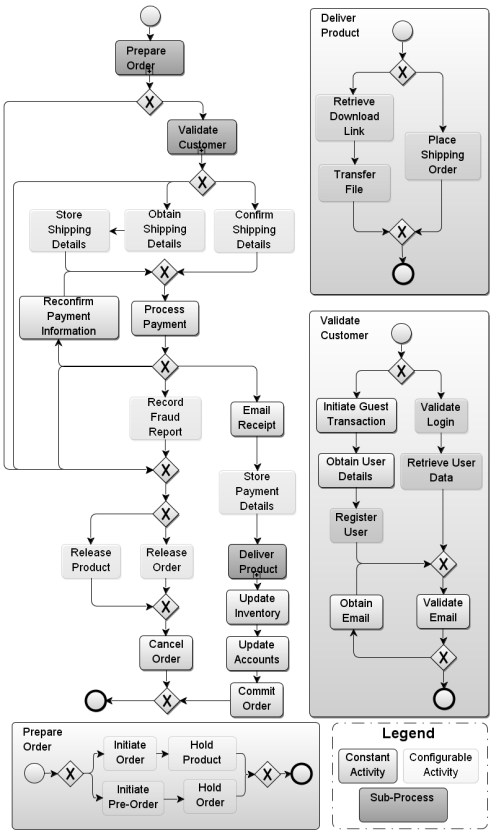


Fig. 4. BPMN model of the configurable checkout BPaaS

TABLE 1
Configurable resources for the BPaaS

Resources	Activities
Private Inventory System, Microguru	Update Inventory, Hold Order, Hold Product, Release Order, Release Product
International Shipping, Toll Priority, AusPost, Client Notification	Place Shipping Order
Provider Storage, Private Customer Repository	Store Payment Details, Validate Login, Confirm Shipping Details, Register User, Store Shipping Details
Private Accounting System, SaaSu	Update Accounts
External Cloud Storage, External Server, Provider Storage	Retrieve Download Link
FTP, FTPS	Transfer File

2.2 Modeling Activities and Control Flow

We use BPMN for modeling activities and control flow, as it formalizes the BPaaS structure while remaining easily readable for clients (see Figure 4 for checkout BPaaS). Furthermore, BPMN is a widely used notation for formalizing and executing business processes, which increases the potential client and provider base of our approach. Configuring BPMN is also less complex than alternatives such as C-EPC, as the alternation events and functions does not need to be maintained [16].

Numerous activities in the checkout BPaaS utilize configurable resources and data objects, which are shown in Tables 1 and 2 respectively. The configurable

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

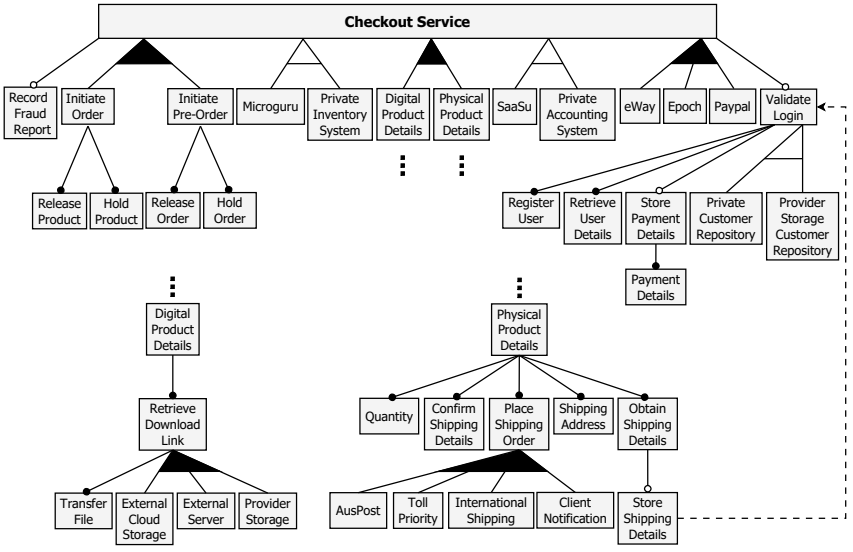


Fig. 3. Feature model of the domain constraints of our example BPaaS

TABLE 2
Configurable data objects for the BPaaS

Data Objects	Activities
Physical Product Details, Digital Product Details, Quantity	Initiate Order, Initiate Pre-Order
Shipping Address	Store Shipping Details, Place Shipping Order, Obtain User Details, Retrieve User Data, Obtain Shipping Details
Payment Details	Process Payment, Retrieve User Data, Obtain User Details, Reconfirm Payment Information

resources include Microguru[9] for inventory management, SaaSu[10] for accounting, and FTP or FTPS for digital product file transfer. Cloud storage is offered by the provider for a customer repository and digital product hosting. Data object configurability includes physical and digital product details, enabling product quantities, and payment and shipping details.

2.3 Modeling Transactional Requirements

While BPMN provides lifecycle statecharts that represent the transactional state of individual activities, a view of the transactional state of the entire process is necessary for verification against process-level transactional requirements. Such requirements include specifying the activities *critical* for successful execution, *necessary* activities to execute prior to aborting, or *requirements* for valid process compensation. We adapt the separated behavior model of our previous work in transactional Web service compositions [19], and use a *transactional behavior model* to represent the global transactional state of the process.

The transactional behavior model of the BPaaS is represented using a statechart, as shown in Figure 5. This model contains various transactional states the BPaaS can be in at a given point, from prior to activation by a client (`Not Activated`), to termination

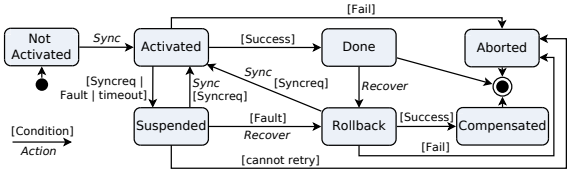


Fig. 5. Transactional behavior model of the BPaaS

through the `Done`, `Aborted`, or `Compensated` states. A `Compensated` state occurs after the effect of an operation is undone through a successful `Rollback`.

Cloud service providers can indicate changes in the transactional state of the BPaaS by modeling *inter-behavior messages* [19] between transactional behavior states and BPMN activities. These messages are exchanged between the transactional and BPMN activity models and are used for their coordination and to facilitate transactional behavior verification:

- *Sync* to trigger an activity execution,
- *Recover* to initiate activities for fault recovery,
- *Delay* to indicate that an unacceptable delay has occurred during execution,
- *Ping* to test the liveness of an activity,
- *Success* to commit a successful execution of the BPaaS,
- *Fail* to abort execution,
- *Syncreq* to request a *Sync* message for retrial following a fault or recovery,
- *Fault* to indicate the presence of a fault that requires recovery, and
- *Ack* to acknowledge a received *Ping* message.

Figure 6 shows an example of how the control behavior model can direct and communicate with the checkout BPMN using inter-behavior messages. In this conversation session, the process becomes suspended after processing payment fails, but the process is able to commit successfully after the user is asked to reconfirm their payment information.

9. http://www.microguru.com/
10. http://www.saasu.com/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

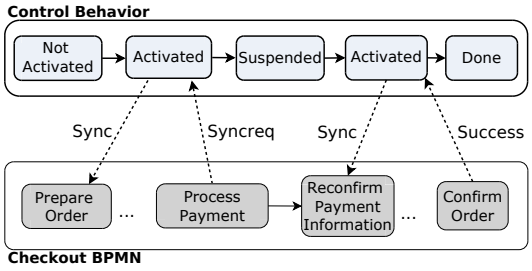


Fig. 6. Inter-behavior messages enabling communication between BPMN and transactional behavior models

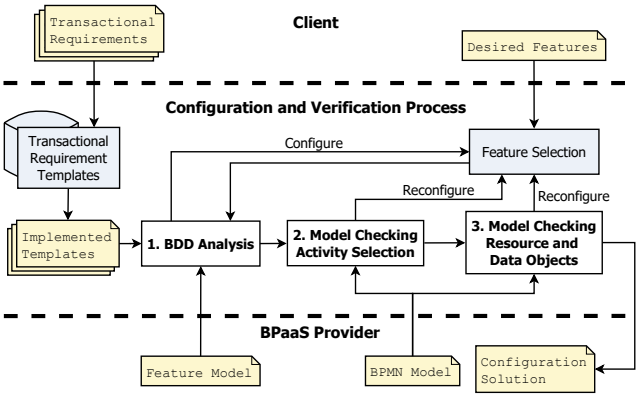


Fig. 7. Overview of the BPaaS configuration and verification process

Transactional requirements can be used by clients to ensure that provisioned BPaaS conform to their expected behavior and internal business rules. These requirements can include specifying satisfactory recovery operations for certain critical activities, conditions for compensating the process, or valid states for aborting execution.

We adapt our work in verifying Web service compositions against developer transactional requirements [19]. This approach allows common or useful transactional requirements to be formalized using *temporal logic templates*, in order to reduce human error and effort. Our proposed template set is divided into *component-level* and *composition-level*, as shown in Table 3. Component-level templates are used for requirements specific to individual activities, while composition-level templates apply to the transactional behavior of the entire BPaaS.

3 Configuration and Verification

In this section, we propose a BPaaS configuration process that applies formal methods to ensure that i) the configuration is valid with respect to provider domain constraints, and ii) the process satisfies transactional requirements drawn from the business rules of the client. First, we provide an overview of the process which guides clients through BPaaS configuration, then we provide details on how Binary Decision Diagram (BDD) analysis and model checking are used at certain steps.

The structure of our configuration and verification process is shown in Figure 7. The aim of this process is to produce a *configuration solution* that satisfies transactional requirements provided by the client while respecting domain constraints. Our configuration process applies formal methods and simple client interaction to identify a configuration that i) is valid with respect to domain constraints, and ii) conforms to the client transactional requirements set. This increases client trust that the service will behave in a manner consistent with internal business policies and requirements, without having to perform their own analysis of the service behavior. The BPaaS client provides the following input:

- *Transactional Requirements:* A specification on the fine-detailed transactional behavior that they require the business process to conform to.
- *Desired Features:* Additional configurable activities, resources, and data objects.

The service provider provides the following input:

- *Feature Model:* A formalization of the domain constraints for valid BPaaS configurations.
- *BPMN Model:* A BPMN model of the complete configurable BPaaS with associated resources, data objects, and transactional behavior model.

Our configuration process uses a series of tasks and formal methods, as shown in Figure 7:

- *Transactional Requirement Templates:* The templates shown in Table 3 for specifying transactional requirements.
- *Implemented Templates:* The requirements implemented using the templates and mapped to concrete temporal logic properties.
- *Feature Selection:* A task where the client selects configurable features of the BPaaS from the feature model. This is used to add features to the configuration, or adjust the BPaaS behavior following failed verification.
- *BDD Analysis:* Verifies that the domain constraints are not violated, given a feature model and a selection of features.
- *Model Checking Activity Selection:* Formally verifies the activity structure of the BPaaS against the formalized transactional requirements, with the exception of requirements with resource and data object concerns.
- *Model Checking Resources and Data Objects:* Formally verifies the complete BPaaS model against the transactional requirements with resource or data object concerns.

3.1 BDD Analysis

The first verification step in our approach identifies all features required for the client transactional require-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

TABLE 3
Component-level and Process-level Transactional Requirement Templates

Type	Template	Description
Component-level	`Compensate Failure`	Specifies an activity and a condition. The failure of the activity requires the condition to be satisfied in the future, in order to recover from the failure.
	`Compensate Success`	Specifies an activity, and a condition that semantically undoes the effect of the completed activity. When the execution of the process must be aborted or compensated, the nominated condition must be satisfied to confirm that the activity has been undone.
	`Alternative`	Following the failure of an activity, one or several alternative activities are considered acceptable replacements.
	`NonRetriable`	Following failure of an activity, retrial is either not possible, or the client is not interested in it.
	`RetriablePivot`	Specifies an activity that may be retried, but not undone. Following its successful completion, the service must commit.
	`NonRetriable Pivot`	Specifies an activity that may not be retried or undone. The service must commit or abort depending on the activity's success.
Process-level	`Transactional StateCritical`	A condition that must be satisfied prior to entering a certain transactional state.
	`Transactional StateTrigger`	A condition that must trigger the entering of a transactional state in the future.
	`Transactional StateReachable`	A condition that indicates that a transactional state is reachable.
	`Transactional StateUnreachable`	A condition that indicates that a transactional state should not be reachable in the future.
	`Compensation`	Specifies a condition that must be met during any compensation process. A compensation process is an activity or series of activities to undo an execution of the service that has committed.
	`Conditional Compensation`	Specifies a condition for compensation and a condition for execution, such that the compensation condition only applies when the execution condition has been satisfied previously.

TABLE 4
Propositional logic representations of the feature model constraints of Figure 2

Constraint	Propositional Logic
Mandatory	$A \leftrightarrow B$
Optional	$B \rightarrow A$
OR	$A \leftrightarrow (B1 \vee B2)$
XOR	$(B1 \leftrightarrow (\neg B2 \wedge A)) \wedge (B2 \leftrightarrow (\neg B1 \wedge A))$
Implication	$A \rightarrow B$
Exclusion	$\neg(A \wedge B)$

ments, and determines whether they can all be selected while satisfying the feature model constraints. This implies that at least one valid configuration must exist using the activities, resources, and data objects specified in the requirements set, or extra features required by the client. We employ BDD-based analysis, which has been proven as an effective method for determining feature model satisfiability [21].

A BDD is an acyclic graph visualization of a propositional logic formula. Variables of the formula are represented as nodes with two outgoing branches, indicating their true or false assignment. The graph is constructed in such a way that each complete path from head node to terminal node represents the assignment of boolean variables. All paths terminate at a final true or false node, which determines whether the variable assignments of that path satisfy the propositional logic formula.

Our analysis transforms the feature model with selections into a propositional logic formula, then into a BDD which can be checked for satisfiability. A feature model can be transformed into a propositional logic formula according to the constraint conversions in Table 4. We use the JDD library[11] to automatically construct BDDs from these propositional logic formulas to check their satisfiability.

Figure 8 shows an example of the transformation

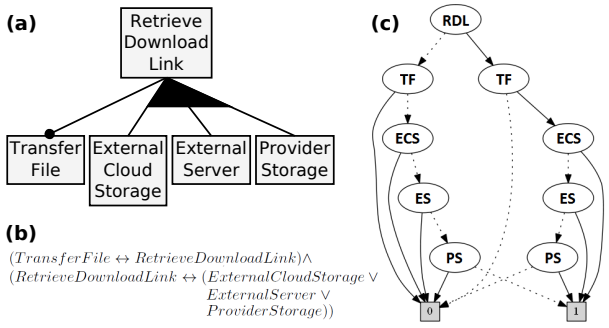


Fig. 8. A selection of domain constraints as a feature model (a), propositional logic (b), and a BDD (c)

steps using a part of the checkout feature model. The BDD in Figure 8(c) uses acronyms of feature names for space considerations, and *0* and `1` nodes for the *false* and *true* respectively. A solid line from a node indicates true assignment, while a dashed line indicates false. By traversing solid lines from selected features, and dashed lines from unselected features, the `1` and *0* nodes indicate whether the preceding path was a valid or invalid selection. The steps in the BDD analysis module in our BPaaS configuration process are:

1) Select all features included in the transactional requirements set,
2) Select any extra features specified by the client,
3) Automatically select all target features of mandatory relationships from those already selected,
4) Verify that all XOR relations with a selected parent feature have exactly one feature selected,
5) Verify that all OR relations with a selected parent have one or more features selected,
6) Construct propositional logic property from depth-first traversal of feature model,
7) Append selections to property with $\vee$ operators,

11. http://javaddlib.sourceforge.net/jdd/index.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

8) Construct Binary Decision Diagram from property using JDD library,
9) Verify that BDD has a reachable *true* node.

BDD analysis is able to efficiently solve satisfiability problems, but the size of the BDD dependent on variable ordering in the underlying propositional logic property. While finding the most efficient ordering is an NP-complete problem [21], ordering variables from a depth-first traversal of the feature model has been approved as an effective strategy [22]. Therefore, we adopt this approach in our work when transforming the feature model to propositional logic.

3.2 Model Checking

Model checking [23] is a formal method that exhaustively verifies that the behavior of a given model conforms to a set of properties. We use the NuSMV model checker [24] to verify BPaaS configurations against the temporal logic forms of conversation rules and transactional requirements. Our verification uses the symbolic BDD-based model checking feature of NuSMV. This constructs BDDs from the input model to apply several verification techniques, including fair CTL model checking and LTL model checking (by an algorithm that reduces the problem to CTL model checking). We use NuSMV as it provides support for properties formulated in both LTL and CTL.

As model checking is an exhaustive technique, it is subject to the state space explosion problem, in which the state space of the system under verification increases exponentially in relation to the number of processes and variables. This greatly impacts verification time and feasibility for complex models. Therefore, prior to verifying BPaaS configurations, we aim to reduce their state space as much as possible. We employ two approaches to curb the impact of state space explosion during BPaaS configuration:

- Applying model checking in *two phases*, reducing the complexity and size of the model and transactional requirements for each phase. Each phase aims to verify a different configuration perspective, namely, *activity selection*, and *resource and data object selection*. These phases also allow the client to focus on each perspective separately. This simplifies the configuration process, especially when configuring a service with a large number of features: if a requirement has no specification of resources and data objects, it only needs verification during activity selection.
- Reducing state space prior to model checking by *transforming the model into a minimal Kripke structure* [25]. Kripke structures are concise representations of system events and are commonly used in formal verification. A Kripke structure is a finite-state system model with a directed graph structure, where each node represents a system state where one or more properties are satisfied.

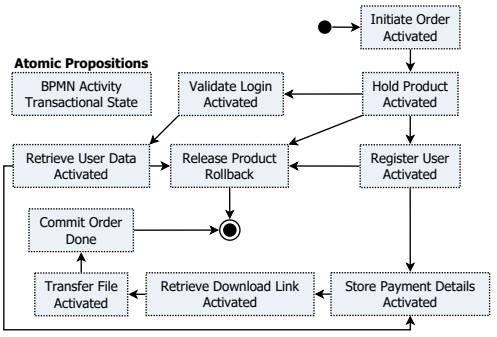


Fig. 9. Kripke structure example for verifying the activity selection

We formally define a Kripke structure as $\mathcal{K} = \langle \mathcal{S}^k, \mathcal{T}^k, \mathcal{L} \rangle$, where $\mathcal{S}_k$ is a finite set of states, $\mathcal{I} \subseteq \mathcal{S}_k$ is the set of initial states, $\mathcal{T}_k \subseteq \mathcal{S}_k \times \mathcal{S}_k$ is the transition function, and $\mathcal{L}$ is the labelling function that assigns *atomic propositions* to each state. Atomic propositions are the unique set of properties that are true for each state. For transactional requirement verification, the set of complete atomic propositions AP contains the activities, resources, data objects, and transactional states included in the implemented templates provided by the client. Figure 9 shows the Kripke structure generated to verify the single requirement that `Retrieve User Data` or `Register User` is necessary before entering the `Done` transactional state. The atomic propositions included in this Kripke structure contain the activities and transactional state necessary for verifying the requirement.

The algorithm to generate these minimal Kripke structures is adapted from our previous work in verifying transactional Web service compositions [19]. Kripke structures are generated by i) identifying the atomic propositions that must be included in the structure, and ii) traversing the BPaaS model in a depth-first order, constructing the structure as those properties are encountered. All configurable features included in the model are also included in the Kripke structure, in order to help the client revise violating behavior with reconfiguration.

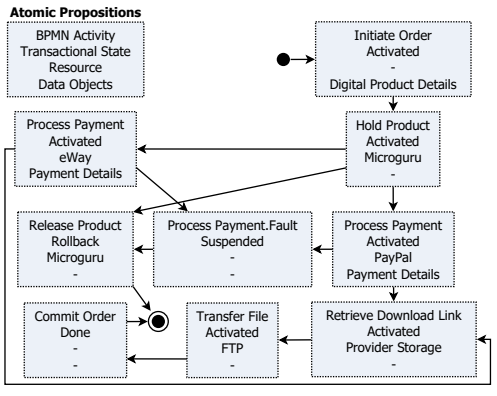


Fig. 10. Kripke structure example for verifying resource and data object selection

3.2.1 Activity Selection

In this phase, if a requirement specifies the inclusion of resources or data objects for an activity, only the presence of the activity in the configured process is verified. This reduces the state space of the process model verified by the model checker, simplifies the properties, and allows the client to focus primarily on activity configuration. To verify activity selection against transactional requirements, the atomic propositions of interest include configurable activities selected from the feature model, and other activities or transactional behavior states used by the implemented set of temporal logic templates.

Model checking verifies the Kripke structure against the temporal logic forms of the client's transactional requirements. If a violation is found, the client must reconfigure the service by interpreting the model checking output; otherwise, the second model checking phase can be undertaken.

3.2.2 Resource and Data Object Selection

In this model checking phase, only transactional requirements with resources or data objects in their specification need to be verified. All other requirements have been ensured in the activity selection phase. This phase is necessary because an activity can have more than one resource provisioned, in order to defer the selection to the user at runtime. For example, a configuration may include several resources for the `Process Payment` activity, as shown in Figure 10. When verifying the activity selection, `Process Payment` will be treated as a single state. However, when considering resources, it becomes several states in an XOR structure, as the choice of resource is deferred to runtime. Data objects are assumed to apply to every runtime instance and do not require process restructuring during verification. If a violation is found, the client must reconfigure the service by interpreting the model checking output.

4 Experimental Evaluation

We have implemented our BPaaS configuration process in a prototype toolset. In this section, we provide an overview of the implementation, before evaluating our approach with configuration scenarios and an extensive performance analysis.

4.1 Implementation

Figure 11 shows the architecture of our prototype tool for BPaaS configuration. The architecture comprises of five modules, which serve the following purposes:

- *Client Interface:* This provides an interface for BPaaS clients to configure services according to our configuration process.
- *Configuration Controller:* This controls the steps of the configuration process as shown in Figure 7. It also handles configuration of BPMN models and feature model interpretation.
- *Verification Controller:* The steps to implement temporal logic templates and reduce the BPaaS model to a minimal Kripke structure are handled by this module.
- *BDD Analysis:* The JDD library is used by an interface in this module to verify that a selection of features does not violate the domain constraints.
- *Model Checking:* This is a wrapper for the NuSMV model checker that handles input parsing, invocation, and output interpretation.

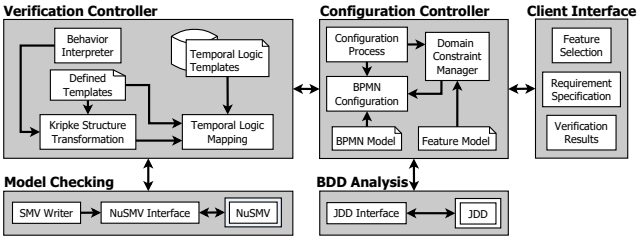


Fig. 11. Overview of the implementation architecture for our BPaaS configuration process

4.2 Validation Scenarios

To validate our BPaaS configuration approach, we demonstrate two scenarios using the Web store checkout service. These scenarios show two clients with quite different transactional and configuration requirements, to demonstrate how our configuration process is suitable for highly configurable BPaaS.

Scenario A: *The client is a medium-sized business looking to expand into online sales with a Web store. The business offers physical products only, and does not do pre-orders. Inventory and accounting are to be managed with SaaS (see Figure 1), but the client has an existing customer repository that will be used. Payment may be made with Paypal and eWay services. The business is looking to outsource the checkout service to an external SaaS provider (SaaS 3 in Figure 1).*

This client has provided 8 transactional requirements that the checkout service must satisfy:

- SA1: `Release Product` is necessary prior to `Aborted` after `Initiate Order` with `Physical Product Details`.
- SA2: `Place Shipping Order` should always lead to `Done`.
- SA3: `Update Accounts` using `SaaSu` is necessary prior to `Done`.
- SA4: Once the activities `Update Accounts`, `Place Shipping Order`, or `Update Inventory` have completed, the process cannot abort.
- SA5: `Obtain Shipping Details` is necessary prior to `Done`.
- SA6: `Reconfirm Payment Information` or `Cancel Order` must be executed following the failure of `Process Payment`.

```
-- specification  G (kstate = Done_CommitOrder ->  O kstate =
Activated_ObtainShippingDetails_nil)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  kstate = initial_state
-> State: 1.2 <-
  kstate = Activated_InitiateOrder
-> State: 1.3 <-
  kstate = Activated_RegisterUser
-> State: 1.4 <-
  kstate = Activated_ConfirmShippingDetails
-> State: 1.5 <-
  kstate = Activated_ProcessPayment
-> State: 1.6 <-
  kstate = Activated_PlaceShippingOrder
-> State: 1.7 <-
  kstate = Done_CommitOrder
-- Loop starts here
-> State: 1.8 <-
  kstate = final_state
```

Fig. 12. Violating stack trace produced by NuSMV

SA7: `Process Payment` using `Paypal` cannot be undone.

SA8: `Process Payment` using `eWay` cannot be undone.

The BDD generated from the domain constraints of Figure 3 and the configurable activities, resources, and data objects included in these requirements indicate that these features selections are valid. In addition to the features specified in these requirements, the client also selects the additional features `AusPost`, `International Shipping` for handling shipping orders, and the `Confirm Shipping Details` activity to streamline the process for recurring customers. BDD analysis confirms that these selections satisfy the domain constraints. It should be noted that the size and complexity of the BDD generated for this analysis is too large to be included in this paper.

During the activity selection model checking phase, NuSMV identified a Kripke stack trace violating `SA5`, shown in Figure 12. From analyzing the stack trace and the BPMN structure in Figure 4, it can be determined that the inclusion of `Confirm Shipping Details` is enabling `Obtain Shipping Details` to be bypassed, thereby violating the requirement that it is critical for successful execution. In order to complete this verification step successfully, the client can i) remove `Confirm Shipping Details` from the configuration, or ii) revise the violating requirement to include it. Following either of these revisions, both model checking phases are successful and a configuration solution is identified.

Scenario B: *An online music store offers digital downloads, CDs, and merchandise of local artists, with shipping currently restricted to domestic customers. The checkout process must be able to handle pre-order sales of upcoming releases. Payment must be handled through the owner's Paypal account, who also maintains customer details, accounting, and inventory.*

This client provides the following transactional requirements for the BPaaS:

SB1: For sales that are not pre-orders, `Transfer File` or `Place Shipping Order` must be successful before the process commits.

SB2: A receipt must be sent to customers following every successful sale.

SB3: Customers must be able to re-enter their `Paypal` details and retry if payment is initially unsuccessful.

SB4: `Process Payment` cannot be undone.

BDD analysis conducted of the configurable features included in this requirement set confirms that they are valid with respect to the domain constraints of the service provider. In addition, the client selects `Initiate Pre-Order`, `Store Payment Details` for configurable activities, `Private Inventory System`, `Private Accounting System`, `Private Customer Repository`, `Provider Storage`, and `AusPost` for resources, and `Digital Product Details` and `Physical Product Details` for data objects.

Once these features are added, the constructed BDD is not satisfiable, as this specific selection of features violates the domain constraints. From analyzing the feature model in Figure 3, we can see that the client has omitted `Validate Login`, but has selected several features that are directly dependent on it. The client plans to handle all customer details management internally, but the BPaaS requires the security of a login procedure in order to conduct sales. To satisfy this constraint, the client removes integration with a `Private Customer Repository` and the `Store Payment Details` activity from the configuration. This activity and resource will be managed completely internally, due to a manageable level of anticipated customers. BPaaS running and provisioning costs may also be reduced if a smaller number of features are included.

After removing those two features from the configuration, BDD analysis passes successfully. Next, the configuration is verified against the transactional requirements across our two model checking phases. Both phases are successful, confirming that the configured BPaaS model satisfies the domain constraints and transactional requirement set.

Scenario C: *A small software development studio has a range of tools to offer consumers through a Web store interface. The products offered by the client are all in the form of digital downloads. Payment must be made using credit card. The client already has an accounting system in place, while inventory management is not required for their exclusive focus on digital products. All sales will be processed as guest transactions.*

The requirements obtained from the client are:

SC1: Customer email addresses must be obtained and verified for the store mailing list.

SC2: Once `Process Payment` has successfully completed, the sale must be finalized.

SC3: Every completed sale must deliver a software product to the customer from an `External Cloud Storage`.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

In addition to the activities obtained from the requirements set, the client nominates `Record Fraud Report` and `Initiate Order`. The configurable resources and data objects included are `Digital Product Details`, `SaaSu`, `External Cloud Storage`, and `Epoch`. A concern for this client is that the `Update Inventory` activity is not configurable, despite not being relevant to their requirements. Provisioning a BPaaS that uses excessive activities or resources outside the client's requirements may have an impact on the overall service performance or cost. As the domain constraints are under the control of the provider, it is the client who must decide if this risk is tolerable for their operations.

If the client persists with the service, by selecting either the `Microguru` or `Private Inventory System` resources, the configuration is confirmed to be valid by the BDD analysis. Furthermore, both model checking phases successfully guarantee their requirements set.

4.3 Performance Analysis

There are several reasons that our configuration process must be able to handle large and complex scenarios in an efficient way. Firstly, the impact state space explosion has on model checking performance is exponential as the size of the model increases. Clients may also have large and complex sets of transactional requirements to be verified. Furthermore, model checking may need to be applied by the client several times, if a configuration solution is difficult to obtain. Therefore, long verification times are likely to compound and become a bigger problem for clients.

4.3.1 Temporal Logic Templates

The first step in our performance analysis is to verify the impact each individual temporal logic template has on model checking performance. Table 5 shows the verification time for minimal Kripke structures, as generated by our Kripke structure reduction algorithm, against one requirement specified using each template. The *Scope* variable required by every template is set to *Global* in each test in order to obtain an even comparison. These results indicate that the templates with the greatest model checking performance demand are `CompensateSuccess`, `Alternative`, `NonRetriablePivot`, and `Compensation`. Our evaluation utilizes these templates in an even ratio.

4.3.2 BPaaS Configuration

We perform two series of performance verification tests for our BPaaS configuration process. The first set of tests aims to validate the performance benefit of our approach using a straightforward model with smaller requirements sets. The configured BPaaS model contains a total of 100 activities (30 configurable), and a combined total of 30 configurable resources and data objects. This model is a BPaaS that has already been configured through feature selection, rather than a configurable BPaaS with 100 total *possible* activities. Using this model, we compare our multi-step model checking approach against a single step model checking approach that does not apply any state space reduction measures. When verifying without state space reduction, the NuSMV input is manually written based on a complete implementation of the model.

We perform 10 tests, verifying the configured BPaaS model given sets of requirements from sizes 10 to 100. The requirements use the same four templates that are used in the previous section. Figure 13 plots the verification times of both the reduced and unreduced models. In the case of unreduced, the verification time is determined from the sum of both model checking phases, and the applications of the *state space reduction* (SSR) algorithm applied at each phase. These results indicate that our approach provides a significant performance benefit in verification time, and that the benefit becomes greater as the requirements sets become larger.

TABLE 5
NuSMV execution times for individual templates

Template	NuSMV (ms)
`CompensateFailure`	7.3
`CompensateSuccess`	7.8
`Alternative`	7.4
`NonRetriable`	7.1
`RetriablePivot`	7.2
`NonRetriablePivot`	7.5
`ControlStateCritical`	7.1
`ControlStateTrigger`	7.1
`ControlStateReachable`	6.5
`ControlStateUnreachable`	6.4
`Compensation`	7.4
`ConditionalCompensation`	7.3

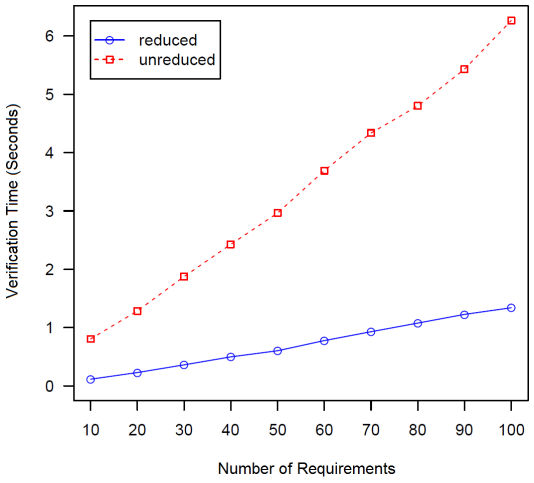


Fig. 13. Verification times during configuration with and without reduction for 10 to 100 requirements

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2667662, IEEE Transactions on Services Computing

TABLE 6
Verification time (in seconds) of increasingly complex configuration scenarios

BPaaS Configuration				NuSMV with reduction					NuSMV with
Act.	Config. Act.	RDO	Reqs	Act. SSR	Act. NuSMV	RDO SSR	RDO NuSMV	Total	no reduction
100	30	30	25	0.007	0.141	0.006	0.107	**0.261**	**1.520**
			50	0.007	0.308	0.008	0.280	**0.603**	**2.963**
			100	0.008	0.725	0.008	0.599	**1.340**	**6.264**
200	60	60	100	0.015	0.835	0.015	0.715	**1.580**	**13.512**
			150	0.022	1.324	0.017	0.963	**2.326**	**20.164**
			200	0.023	1.921	0.021	1.591	**3.556**	**28.365**
300	90	90	200	0.027	2.116	0.022	1.782	**3.947**	**46.538**
			250	0.027	2.775	0.022	2.290	**5.114**	**60.972**
			300	0.030	3.468	0.026	2.788	**6.312**	**73.553**
500	150	150	300	0.053	4.246	0.042	3.391	**7.732**	**191.103**
			400	0.053	5.861	0.045	4.636	**10.595**	**256.413**
			500	0.057	7.643	0.047	5.963	**13.710**	**323.896**

4.4 Discussion

Our work is a first step towards a comprehensive BPaaS management framework that is capabale of responding to changes in the environment and managing client transactional requirements. However, several limitations still exist. Firstly, we recognize that the interface exposed to the client might have significant complexities, which are not easy to tackle by non-system experts. An interface to hide the details and complexity of the service aside from the transactional requirement specification and feature selection would ensure that this work is widely used. As BPaaS is still an emerging technology, this framework especially one that enables clients to ensure complex transactional requirements, would be a significant contribution to the field. Existing approaches that manage cloud service evolution focus on requirements such as process structure fairness [6], and preserving elasticity [11] and multi-tenancy [26]. To the best of our knowledge, complex properties such as transactional requirements are yet to receive attention in this area. The work presented in this article will hopefully provide avenues for addressing a host of opportunities present in the early stages of BPaaS research.

5 Related Work

There is an increasing research interest in related areas such as configurable cloud service applications [26][27], and configurable or adaptive business process models [14][16][17]. Based on our analysis of related work as well as the various uses of BPaaS as outlined by the scenarios in Section 4, we identify a set of criteria as shown in Table 7, and group them in two categories, namely, *Support*, focused on the modeling of the business process, and *Correctness*, focused on the verification of process correctness.

The *Support* criteria is related to the business process expressiveness enabled by the approach. The *Process Formalism* criterion identifies how the approach expresses business processes. If business process structures are not explicitly incorporated in the approach, the value is left blank. The next criteria specify whether it is capable of modeling resource and data and configurability. Finally, the means of expressing domain constraints are identified for comparison. *Correctness Criteria* identifies the properties that are ensured during the configuration or adaptation approach. Process model criteria refers to structural or behavioral correctness of the process model, such as soundness [28] or syntactical correctness [12]. Some approaches may also analyze the configurability of the model to identify issues such as contradictions or circular dependencies [29]. The final criterion identifies the client requirements that are input into the process, such as selections of features [5][30], or more complex behavioral requirements [15].

TABLE 7
Comparison criteria overview

Category	Criteria	Values
Support	Process Formalism	Petri-nets, BPMN, statecharts
	Resources	$\surd$ or -
	Data	$\surd$ or -
	Domain Constraints	Feature model, OVM, hierarchy model
Correctness Criteria	Process Model	Syntax, Soundness, Deadlock-freedom
	Configurability	Circular dependency-free, contradiction-free
	Client Requirements	Feature selections, business rules, QoS parameters

Approaches to managing cloud service configuration has so far mostly focused on SaaS [5][26][30]. These approaches share many concerns with BPaaS configuration, such as managing domain constraints, and eliciting requirements from clients. The configurable features of SaaS can include user interface, program structure, data, access control, and other properties [26][30]. Several approaches have addressed the problem of *evolving multi-tenant* SaaS at runtime [5][27][31]. Multi-tenant services are able to handle several clients with a single software instance, in a way that prevents any interference between the specific configurations of each client. As new clients provision the service at runtime, the service must evolve its behavior to meet new requirements, while still supporting the existing clients.

While configurable BPaaS is still a new area of research, work in related domains such as config-

TABLE 8
Support criteria for comparing work related to BPaaS configuration

Approach	Process Formalism	Resources	Data	Domain Constraints
Mietzner et al. [26]	-	√	√	OVM
Lizhen et al. [30]	-	√	√	Metagraph relationships
Schoreter et al. [7]	-	√	√	QCL Contracts
Banerjee [31]	-	√	√	Variation pre and post-conditions
Kumara et al. [5]	-	√	√	Feature model
Schroeter et al. [27]	-	√	√	Extended feature model
Mendling et al. [12]	C-EPC	-	-	XPath Statements
van der Aalst et al. [32]	Workflow nets	-	-	-
van der Aalst et al. [13]	Workflow nets, C-EPC	-	-	-
Kumar and Yao [15]	WS-BPEL	√	√	If-then statements
La Rosa et al. [33][16]	C-iEPC	√	√	Hierarchy model
La Rosa et al. [29]	-	-	-	Configuration Model
Wang et al. [34]	WS-BPEL	√	√	-
van der Aalst [35]	C-nets	-	-	C-net bindings
Tsai and Sun [36]	Customizable workflow	√	√	OVM
Liu et al. [6]	Petri-nets	-	-	-
van Dongen et al. [28]	EPC, Petri-nets	-	-	-
Gröner et al. [14]	BPMN	√	√	Feature Model
van der Aalst et al. [17]	Petri-nets	-	-	-
Jiang et al. [37]	Dependency structures	-	-	-
Hallerbach et al. [38]	Workflow model	-	-	Basic logic
Gottschalk et al. [39]	LTS, C-EPC	-	-	-

urable business processes [15][16][17] and reference models [13][14] has been ongoing for several years. Most approaches are enabled during the configuration of business processes [12][13][32], while others verify configurable business process models against correctness properties [14][28], or obtain a set of all valid configurations for clients to select from [17][37].

Table 8 shows that among 22 approaches we identified, only 65% apply themselves to business process structures. Furthermore, only one third of those approaches support resource and data configuration. As indicated by the configurable cloud service approaches in our survey, resource and data configurability is an important feature for allowing clients to tailor services towards their requirements. Allowing clients to configure resources can have an impact on the running costs of the service, in addition to increasing the potential market with greater configurability. Data object configuration allows clients to make the service more similar to existing or expected business practises. Therefore, we consider it important for research into BPaaS configuration to consider configuration from these perspectives.

Also, as the *Client Requirements* field in Table 9 indicates, most approaches only support simple requirements such as selecting and removing features. Feature selections are all the client provides in 26% percent of our surveyed approaches, while other basic binary selections, such as blocking or hiding activities, make up a further 39%. Other approaches provide some business rule support [15][34], such as basic if-then conditions. To the best of our knowledge, transactional requirements important to clients, such as those supported by our template set, are not yet supported by any business process configuration method, and this is one of the major contributions of this work compared to existing works.

We also identify that little research so far has addressed BPaaS explicitly. Approaches such as [6] target their configuration method towards SaaS that has a business process structure of activities, while some configurable business process methods consider variability of resources and data associated with activities. However, BPaaS has their own unique concerns such as configurable use of third-party services, and the inherent transactional concerns.

Our approach manages BPaaS configuration in a way that addresses the issues identified above. Firstly, our BPaaS model enables configuration from numerous perspectives important to BPaaS clients, namely, activities, resources, and data objects, as shown in the scenario examples. Our configuration method aims to elicit and ensure complex transactional requirements from clients, by adapting the temporal logic template set as shown in Section 4. This is in contrast to the approaches proposed in [17][37] but it has the advantage of a reduced runtime when configuring services with many configuration options and values.

6 Conclusion

The increase in cloud computing adaptations in recent years has produced the concept of *Business Process as a Service* (BPaaS), whereby service providers are able to offer common or proven business processes to clients looking to automate and/or outsource parts of their operations. We address the problem of managing BPaaS configuration in a way to ensure that the resulting service i) is valid with respect to configuration constraints of the provider, and ii) satisfies transactional requirements drawn from the business rules of the client. Our approach utilizes several modelling techniques, including BPMN for business process structure, statecharts for transactional state, feature models for configuration constraints. Using these

TABLE 9
Correctness Criteria for comparing work related to BPaaS configuration

Approach	Process Model	Configurability	Client Requirements
Mietzner et al. [26]	-	-	Feature selections
Lizhen et al. [30]	-	-	Feature selections
Schoreter et al. [7]	-	-	Feature selections
Banerjee [31]	-	-	Required services
Kumara et al. [5]	-	-	Feature selections
Schroeter et al. [27]	-	-	Feature selections
Mendling et al. [12]	Syntax	-	Function and connector variants
van der Aalst et al. [32]	Syntax and Soundness	-	Blocked and hidden activities
van der Aalst et al. [13]	Syntax and Soundness	-	Blocked and hidden activities
Kumar and Yao [15]	Syntax	-	Business rules
La Rosa et al. [33][16]	Syntax	-	Blocked and optional functions, resources, data objects, connectors
La Rosa et al. [29]	-	No circular dependencies or contradictions	Boolean questionnaire answers
Wang et al. [34]	-	-	Set of business policy types
van der Aalst [35]	-	-	Blocked activities
Tsai and Sun [36]	-	-	Structure, GUI, resources, data, QoS
Liu et al. [6]	Fairness, deadlock-free, reachability	-	Adding, deleting, modifying elements
van Dongen et al. [28]	Relaxed soundness	-	Termination states
Gröner et al. [14]	-	Control flow and domain constraint conflicts	Feature selections
van der Aalst et al. [17]	Weak-termination	-	Blocked and hidden activities
Jiang et al. [37]	Deadlock-free	-	Blocked activities
Hallerbach et al. [38]	Soundness	-	Security, maintenance, and workload context variables
Gottschalk et al. [39]	-	-	Blocked, hidden, and optional activities

models, we develop a BPaaS configuration process that applies Binary Decision Diagram (BDD) analysis and model checking. BDD analysis ensures that BPaaS features selected during configuration do not violate the domain constraints of the service provider, while model checking verifies the configured BPaaS against transactional requirements provided by the client. To reduce the impact of state-space explosion, we employ a state-space reduction algorithm and split the model checking into two phases. These phases verify different configuration perspectives separately, and allow for the state space and temporal logic properties to be reduced further. Our performance analysis shows that the proposed configuration method is capable of verifying models with hundreds of activities, resources, data objects, and requirement sets within seconds.

ACKNOWLEDGMENTS

Quan Z. Sheng's work has been partially supported by Australian Research Council (ARC) Discovery Grant DP0878917 and FT140101247. The authors would like to thank the anonymous reviewers for their valuable feedback on this work.

REFERENCES

[1] S. Bouchenak, G. Chockler, H. Chockler, G. Gheorghe, N. Santos, and A. Shraer, "Verifying cloud services: present and future," *SIGOPS Operating Systems Review*, vol. 47, no. 2, 2013.

[2] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud Computing The Business Perspective," *Decision Support Systems*, vol. 51, no. 1, pp. 176–189, 2011.

[3] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud Computing: State-of-the-Art and Research Challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[5] I. Kumara, J. Han, A. Colman, and M. Kapuruge, "Runtime Evolution of Service-Based Multi-Tenant SaaS Applications," in *Service-Oriented Computing*. Springer, 2013, pp. 192–206.

[6] Y. Liu, B. Zhang, G. Liu, M. Zhang, and J. Na, "Evolving SaaS Based on Reflective Petri Nets," in *Proceedings of the 7th Workshop on Reflection, AOP and Meta-Data for Software Evolution*. ACM, 2010, pp. 7:1–7:4. [Online]. Available: http://doi.acm.org/10.1145/1890683.1890690

[7] J. Schroeter, S. Cech, S. Götz, C. Wilke, and U. Aßmann, "Towards Modeling a Variable Architecture for Multi-Tenant SaaS-Applications," in *Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems*. ACM, 2012, pp. 111–120.

[8] R. Accorsi, "Business Process as a Service: Chances for Remote Auditing," in *The 35th Annual Computer Software and Applications Conference Workshop*. IEEE, 2011, pp. 398–403.

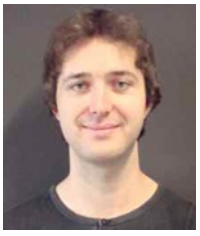
[9] S. Bourne, C. Szabo, and Q. Sheng, "Managing Configurable Business Process as a Service to Satisfy Client Transactional Requirements," in *Proceedings of the 11th International Conference on Services Computing*. IEEE, 2015, pp. 154–161.

[10] T. Lynn, J. Mooney, M. Helfert, D. Corcoran, G. Hunt, L. Van Der Werff, J. Morrison, and P. Healy, "Towards a Framework for Defining and Categorising Business Process-As-A-Service (BPaaS)," in *21st International Product Development Management Conference*, 2014.

[11] M. P. Papazoglou and W. van den Heuvel, "Blueprinting the Cloud," *IEEE Internet Computing*, vol. 15, no. 6, pp. 74–79, 2011.

[12] J. Mendling, J. Recker, M. Rosemann, and W. M. P. van der Aalst, "Generating Correct EPCs from Configured C-EPCs,"

in *Proceedings of the Symposium on Applied Computing*. ACM, 2006, pp. 1505–1510.
[13] W. M. P. van der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. La Rosa, and J. Mendling, "Preserving Correctness During Business Process Model Configuration," *Formal Aspects of Computing*, vol. 22, no. 3-4, pp. 459–482, 2010.
[14] G. Gröner, M. Bošković, F. S. Parreiras, and D. Gašević, "Modeling and Validation of Business Process Families," *Information Systems*, vol. 38, no. 5, pp. 709–726, 2013.
[15] A. Kumar and W. Yao, "Design and Management of Flexible Process Variants using Templates and Rules," *Computers in Industry*, vol. 63, no. 2, pp. 112–130, 2012.
[16] M. La Rosa, M. Dumas, A. H. M. ter Hofstede, J. Mendling, and F. Gottschalk, "Beyond Control-Flow: Extending Business Process Configuration to Roles and Objects," in *Proceedings of the 27th International Conference on Conceptual Modeling*, 2008, pp. 199–215.
[17] W. M. P. van der Aalst, N. Lohmann, M. Rosa, and J. Xu, "Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis," in *Proceedings of the 8th International Conference on Business Process Management (BPM 2010)*, ser. Lecture Notes in Computer Science, vol. 6336, pp. 95–111.
[18] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT press, 2008.
[19] S. Bourne, C. Szabo, and Q. Sheng, "Verifying Transactional Requirements of Web Service Compositions using Temporal Logic Templates," in *Proceedings of the 14th International Conference on Web Information System Engineering*. Springer, 2013.
[20] K. C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, vol. 19, no. 4, 2002.
[21] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated Analysis of Feature Models 20 Years Later: A Literature Review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.
[22] W. Zhang, H. Yan, H. Zhao, and Z. Jin, "A BDD-Based Approach to Verifying Clone-Enabled Feature Models Constraints and Customization," in *High Confidence Software Reuse in Large Systems*. Springer, 2008, pp. 186–199.
[23] E. Clarke, "Model Checking," in *Foundations of Software Technology and Theoretical Computer Science*. Springer, 1997.
[24] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An Opensource Tool for Symbolic Model Checking," in *Computer Aided Verification*. Springer, 2002, pp. 241–268.
[25] S. Kripke, "Semantical Considerations on Modal Logic," *Acta Philosophica Fennica*, vol. 16, no. 1963, pp. 83–94, 1963.
[26] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl, "Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications," in *Proceedings of the ICSE Workshop on Principles of Engineering Service Oriented Systems*. IEEE, 2009, pp. 18–25.
[27] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau, "Dynamic Configuration Management of Cloud-based Applications," in *Proceedings of the 16th International Software Product Line Conference (SPLC 2012)*, pp. 171–178.
[28] B. F. van Dongen, M. H. Jansen-Vullers, H. M. W. Verbeek, and W. M. P. van der Aalst, "Verification of the SAP Reference Models using EPC Reduction, State-Space Analysis, and Invariants," *Computers in Industry*, vol. 58, no. 6, 2007.
[29] M. La Rosa, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, "Questionnaire-Based Variability Modeling for System Configuration," *Software & Systems Modeling*, vol. 8, no. 2, pp. 251–274, 2009.
[30] L. Cui, H. Wang, J. Lin, and H. Pu, "Customization Modeling Based on Metagraph for Multi-Tenant Applications," in *The 5th International Conference on Pervasive Computing and Applications*. IEEE, 2010, pp. 255–260.
[31] A. Banerjee, "A Formal Model for Multi-Tenant Software-as-a-Service in Cloud Computing," in *Proceedings of the 5th ACM COMPUTE Conference: Intelligent & Scalable System Technologies*. ACM, 2012, p. 18.
[32] W. M. P. Van Der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. La Rosa, and J. Mendling, "Correctness-Preserving Configuration of Business Process Models," in *Fundamental Approaches to Software Engineering*. Springer, 2008.
[33] M. La Rosa, M. Dumas, A. H. M. ter Hofstede, and J. Mendling, "Configurable Multi-Perspective Business Process Models," *Information Systems*, vol. 36, no. 2, pp. 313–340, 2011.
[34] M. X. Wang, K. Y. Bandara, and C. Pahl, "Process as a Service Distributed Multi-Tenant Policy-Based Process Runtime Governance," in *The International Conference on Services Computing*. IEEE, 2010, pp. 578–585.
[35] W. M. P. van der Aalst, "Business Process Configuration in the Cloud: How to Support and Analyze Multi-tenant Processes?" in *Proceedings of the 9th European Conference on Web Services*, 2011, pp. 3–10.
[36] W. Tsai and X. Sun, "SaaS Multi-Tenant Application Customization," in *The 7th International Symposium on Service Oriented System Engineering*. IEEE, 2013, pp. 1–12.
[37] J. M. Jiang, S. Zhang, P. Gong, and Z. Hong, "Configuring Business Process Models," *SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–10, 2013.
[38] A. Hallerbach, T. Bauer, and M. Reichert, "Guaranteeing Soundness of Configurable Process Variants in Provop," in *IEEE Conference on Commerce and Enterprise Computing*. IEEE, 2009, pp. 98–105.
[39] F. Gottschalk, W. M. P. van der Aalst, and M. H. Jansen-Vullers, "Configurable Process Models A Foundational Approach," in *Reference Modeling*. Springer, 2007, pp. 59–77.



Scott Bourne received his PhD degree in computer science from the University of Adelaide in the School of Computer Science in 2016. His research interests include service-oriented computing, formal methods, verification, and program analysis. Scott has published in several conferences (e.g., ICSOC, WISE, SCC) and international journals (e.g., Information Sciences). He has recieved a number of awards due to his research work including a Dean's Commendation for Doctoral Thesis Excellence from the University of Adelaide in 2016.

Claudia Szabo is currently a lecturer at School of Computer Science, the University of Adelaide. She is also an Associate Dean (DI) of Faculty of Engineering, Computer and Mathematical Sciences. Claudia received her PhD degree in computer science from National University of Singapore. Her research interests include model-driven engineering, distributed and cloud computing, verification and validation of distributed systems. She is the author of more than 50 publications. She is a member of IEEE.

Quan Z. Sheng is a full professor and Head of Department of Computing, Macquarie University. He received the PhD degree in computer science from the University of New South Wales in 2006. His research interests include service computing, distributed computing, Internet computing, big data analytics, Internet of Things, and Web of Things. He is the recipient of ARC Future Fellowship in 2014, Chris Wallace Award for Outstanding Research Contribution in 2012, and Microsoft Research Fellowship in 2003. He is the author of more than 270 publications. He is a member of ACM and IEEE.