Refactoring Service Interface to Support Artefact-centric Service interaction*

Fuguo Wei, Alistair Barros, and Chun Ouyang
Queensland University of Technology, Brisbane, Australia
{f.wei,alistair.barros,c.ouyang}@qut.edu.au

Abstract. The growth of APIs and Web services on the Internet, especially through larger enterprise systems increasingly being leveraged for Cloud and software-as-a-service opportunities, poses challenges for improving the efficiency of integration with these services. Interfaces of enterprise systems are typically larger, more complex and overloaded, with single operations having multiple data entities and parameter sets, supporting varying requests, and reflecting versioning across different system releases, compared to fine-grained operations of contemporary interfaces. We propose a technique to support the refactoring of service interfaces by deriving business entities and their relationships. In this paper, we focus on the behavioural aspects of service interfaces, aiming to discover the sequential dependencies of operations (otherwise known as protocol extraction) based on the entities and relationships derived. Specifically, we propose heuristics according to these relationships, and in turn, deriving permissible orders in which operations are invoked. As a result of this, service operations can be refactored on business entity CRUD lines, with explicit behavioural protocols as part of an interface definition. This supports flexible service discovery, composition and integration. A prototypical implementation and analysis of existing Web services, including those of commercial logistic systems (Fedex), are used to validate the algorithms proposed through the paper.

Keywords: web service, business entity, service interface synthesis, service interaction

1 Introduction

Conventional service adaptation [1, 2] relies on expert insight of service providers to gain an understanding of service interfaces so that they can be integrated, composed and accessed with external applications. Through insights into the interfaces of the applications intended to interoperate, adapters can be built to support mediation across heterogeneous data types of operations (structural aspects) and the permissible orders in which operations are invoked (behavioural aspects). With the growth of interfaces on the Internet, especially for larger,
enterprise systems from SAP, Oracle, Fedex and the like, classical adaptation and other mechanisms for achieving service integration are time-consuming and costly, due to the size, complexity and overloading of enterprise systems interfaces. The interfaces of larger systems typically suffer from non-modularity due to legacy design, meaning that the operations in the interface are overloaded: they have multiple data entities and related parameter sets in their operations, supporting a variety of different service requests, making it difficult to determine the essential functions supported through the operations. Furthermore, different versions through the same operation compound the challenges of a modular, functional understanding of service interfaces, considering for instance that FedEx Web services have more than 1000 parameters in some of operations, while SAP enterprise services have up to 400 parameters.

This paper contributes to a complementary strategy to conventional service adaptation, whereby knowledge of service interfaces can be unilaterally analysed by service consumers in support of self-learning and self-adaptation with external services. Specifically, we have extended upon previous efforts to analyse interfaces for data type elicitation and data dependencies by automatically analysing service interfaces [3, 4]. These are useful for identifying the focal artefacts of applications, namely the business entities, which forms the basis for the creation of a simplified and fine-grained interface layer, allowing access (create, read, update and delete) operations against individual business entities. Through [5], we have proposed refined insights into the discovery of business entities and their relationships, and shown how these can be used to refactor fine-grained, artefact-centric operations, validated using several existing Web services including those of SAP and FedEx applications.

In this paper, we extend service interface analysis for behavioural aspects, proposing how sequential dependencies of operations can be discovered and used to generate behavioural protocols in service interfaces. The protocols are derived from an understanding of different relationships between business entities. As an example, if the analysis of an operation elicits two business entities, one of which exclusively contains the other (e.g. a line item is exclusively contained in a purchase order), the creation of a line item should be synchronized with the creation of the purchase order. This implies a triggering dependence between the corresponding operations: purchase order creation and line item creation. In all, we consider 3 types of relationships across business entities and propose heuristics for triggering dependencies: exclusive containment, inclusive containment (mandatory and optional) and association. These, in turn, result in different business entity operation invocation dependencies, providing indispensable knowledge for generating behavioural aspects of service interfaces.

The remainder of this paper is structured as follows. Sect. 2 reviews state of the art and this is followed by the elaboration on the key algorithms of the behavioural interface derivation mechanism and the development of detailed insights into its most novel features in Sect. 4. Sect. 5 evaluates the mechanism by experimenting the implemented prototype with a variety of services and reveals some open issues. Finally, Sect. 7 concludes the paper and outlines the future work.
2 Related Work

Various techniques have been proposed over recent years to address challenges of generating service behavioural interfaces. These approaches include static code analysis, semantic ontologies, interaction log mining, and service composition.

Static analysis involves analysing codes of web applications. For example, Lucca et al. [6] proposed a two-phase analysis approach, where source codes (such as HTML tags and PHP programs) and execution of web applications are analysed and observed. However, this proposal relies on the availability of source codes and it also requires a significant amount of human intervention. In addition, the approach is limited to recovering high-level design documentation such as sequence diagrams.

Semantic ontologies have been utilised to annotate service interfaces. As an example of this utilisation, Falk et al. [7] adapted automata learning to the problem of service interaction analysis. This proposal usefully combines automated analysis with semantic ontologies in that it requires semantically annotated interface descriptions showing preconditions and effects as the prerequisite to learn interaction protocols. Also, there are web service Semantics standards such as WSDL-S \(^1\), which are meant to incorporate ontologies into services so that behavioural interfaces can be derived with ease. However, they have not been commonly practised, because they impose a considerable amount of development work on service providers and the maintenance of semantic ontologies requires significant lead times and adoption.

Complementary to semantic techniques, log mining algorithms [8] have been proposed for discovering service protocols (i.e., behavioural interfaces according to our terminology). The mining technique incurs overheads for aggregating logs and can suffer from lack of logs or even missing information in them.

Service composition have been investigated intensively, and the common problem being addressed is in this area is “how to automatically generate a new target service protocol by reusing some existing ones” [9]. However, this technique assumes that the behavioural interfaces of individual services involved in a composition are available.

Another proposal by Bertolino et al. [3] synthesises service behavioural interfaces based on type elicitation and data dependencies between service operations’ input and output parameters. We extend the analysis to derive the central artefact - business entities and their relationships, namely exclusive containment, inclusive containment and association in order to transform service operations into CRUD of business entities. Ultimately, ordering constraints are developed among these operations based on the relationships between entities. Kumaran et al. [4] proposed an approach to transform a process activity based process model to an information-centric one, where life cycles of business entities are incorporated into business process models. This research has demonstrated the importance of modelling a process using information entities (a similar notion to the business entity in our study). This research also proposed a relation between

\(^1\) http://www.w3.org/Submission/WSDL-S/
information entities called domination, which has been adapted in our study to support the derivation of relationships between business entities.

Due to the complexity of multiple service conversation in business networks, there exits not only the fundamental structural and behavioral incompatibilities, but also interaction problems that may lie in the semantic business rules, dynamic conversation constraints, conversation optimization needs, and dynamic business partnership. Barros et al. [10] presents a spectrum of service interaction patterns in business process settings. Interaction patterns discussed in the paper cover both routed patterns and non-routed patterns. However, the paper only abstracts interaction behavior and it has not examined interaction issues such as incompatibilities and problems encountered during interaction optimization in the business network setting. Message correlation patterns are discussed in [11]. Nevertheless, the paper serves as a general message correlation guideline and it does not address uncertainties in business networks.

3 State-based Service interaction Patterns

Pattern 1 (Flexible interaction selection)

Description Interactions with services are selected based their conformance to the state of conversations. Specifically, a service has a desired progress in a conversation through a current and desired state. Several transitions may exist to achieve the desired state. The different transitions allow other services to determine which services and interactions can be used to achieve the desired state. The only constraint is that the interaction protocols with individual services are not violated. For example, in Fig. 1, to transit from the current state to the desired state, there are two (or more than two) transitions (corresponding to 2 interaction sets) proposed. This allows a flexible interaction selection between the interaction sets proposed.

![Fig. 1. An example of flexible interaction selection.](image-url)
Issues/design choices The key issue is alignment of conversation state transitions and service interactions, meaning that state based service behavioural knowledge is required in order. Parts of interaction sets need to be carefully demarcated and associated with state transitions, allowing relevant interaction sets to be selected, to achieve the desired state. Available implementation mechanisms such as events can be used to indicate completion of interaction sets, whereby events correlate (are subscribed by) with transitions. Note, the interaction sets do not need to have full correspondence with protocols, as parts of protocols may be sufficient to fulfill a transition. Of course, all services involved in a conversation would need to have a shared understanding of conversation states and transitions. One way of achieving this is to locally store this knowledge in the messaging systems of individual services. Another approach could be to make use of a central service mediator. The former, peer-to-peer approach, would require services to pass the current and desired states to services they interact with to progress the conversation. The latter, central mediator approach could go a step further and orchestrate conversations, since the conversation state transitions and interaction mappings are available to it.

Examples A shipper requires the booking of carrier services, booking of warehouse storage, customs/quarantine scheduling, and the acquisition of a banking letter of credit. Normally a transportation management service would cost and book these services, one at a time. However, a conversation may be designed whereby a single transition from states “quotation approved” (current state) to “delivery planned” (desired state) to allow interactions with the relevant and available service providers in a network to be undertaken.

Pattern 2 (Flexible interleaved interactions)

Description Interactions with services are interleaved based on their conformance to the state of conversations. Normally, one set of interactions takes place between a service and another service, followed by interactions between one of these services and further service. Since interaction sets can be arbitrarily correlated with transitions, parts of the first set of interactions can take place, interleaved with parts of the second set of interactions, in order to achieve the desired state. Once again, note that the protocols with individual services are not violated, even if parts of their allowed interactions correlate with a transition. As illustrated in Fig. 2, the interaction set 1 is chosen to realize the transition 1 and the interaction set 2 is for the transition 2. It can be seen that some of interactions from interaction set 2 interleave with interactions from the interaction set 1 while the transition 1 is in process.

Issues/design choices The same issues apply as with pattern 3. The crucial issue is that interleaving interactions from otherwise strictly partitioned protocols can result in unsafe executions, e.g. deadlock. If parts of the interactions in a protocol are assigned to one transition, then the other parts should also be correlated to other transitions in a conversation such that a conversation progresses across state transitions without deadlock. Thus, conversations need to be formally verified for safe execution across transitions and interaction sets, to avoid deadlock and non-termination. Another issue is ensuring the internal in-
In the diagram, Service 1, Service 2, and Service 3 are represented. The arrows between the services indicate the direction of data flow.

**Fig. 2.** An example of flexible interleaved interactions.

The integrity of services is not compromised as a result of interleaving, for example the invocation of services in parallel with data flow dependencies can cause data integrity issues in the individual systems supporting the services. This issue is hard to overcome from a purely conversation interaction perspective, but should be taken into account when designing interaction set mappings and state transitions.

**Examples** A shipper requires the booking of carrier services, booking of warehouse storage, customs/quarantine scheduling, and the acquisition of a banking letter of credit. Interaction protocols of carrier services involve quotation and booking steps, which can be lengthy processes. This can delay other services like letter of credit and customs/quarantine scheduling, normally done subsequently. For optimization purposes, a banking letter of credit service could be started in parallel, allowing the payment process to be planned during transportation planning. In turn, when both transportation and payment states are reached, the transportation planning completion state could be automatically reached.

**Pattern 3 (Cancel interactions to previous state)**

**Description** Interactions with one or more services are cancelled to a previous state in a conversation. This is achieved by issuing cancellations for all interactions that are currently taking place or have completed between the current state of a conversation and a previous state. Several sets of interactions may be cancelled across several state transitions, as a result.

**Issues/design choices** The main issue is to ensure that actions for cancellations for interactions are in place. These may involve rollbacks of open interactions or compensations for completed interactions. The execution of cancellations should be in reverse order of the execution of interactions, and the effects across different services should preserve integrity.
Examples One use case for cancellations could be where services have been invoked and provide redundant capabilities, e.g. a prime carrier in a transportation carrier service may provide integrated bookings with other carriers and also warehouse bookings and customs/quarantine scheduling. Depending on how conversations are structured such redundancy can be built in to take advantage of best of breed services in a network. However, the design of cancellations should be factored in, so that such redundancies can be rolled-back, when the best service plan emerges.

Pattern 4 (Replace interactions to future state)

Description Replacing interactions involves canceling interactions to a previous state in a conversation, and executing interactions to a future state (e.g. the current state as the point of cancellation). This allows parts of a conversation with existing services to be replaced. We have discussed cancellations in pattern 6. Replacing services and interactions to a future state would seem to be no different to normal execution of a conversation. However, there is a difference in that any information instrumental to requesting new services etc, should not be lost from previous interactions. They should be stored for invoking new services.

Issues/design choices Replacing interactions with new services requires that information used to invoke previously used services be persisted. This adds to the complexity in individual services and central mediators in storing and retrieving this information, which can be costly in terms of performance.

Examples Due to market demand, a company decides to stop manufacturing the current products and shift to produce another type of goods which is increasingly demanded by the market. Therefore, the purchase service wants to cancel the current supplier service and replace it with another supplier that supplies the materials for the new products.

4 Behavioural Interface Derivation

4.1 Overview of the Approach

We propose a four-step approach for deriving service behavioural interface, which is illustrated in Fig. 3. Given a service specification such as a WSDL file, the first step is to identify business entities and create business entity-based data models capturing service structural interface. This can be achieved by iterating the operations involved in the service and examining all the complex input and output parameters of individual operations. The details of generating such a data model for service syntactical interface can be found in our previous work [5]. The resulting data model is then utilised in the following three steps, which are the primary focus of this paper. In the second step, the operations provided by a service are analysed and grouped into several categories based upon what exactly each operation does to business entities, namely creating, reading, updating, deleting, and associating. Then, the behavioural model for creating a business entity is generated according to a number of rules that are derived based on three types of relationships (i.e., exclusive containment, inclusive containment, and association) among business entities. This model is the key output of the behavioural
interface derivation mechanism. Finally, the notion of state is incorporated into business entities, and a model that reflects a business entity’s life cycle is generated. The resulting behavioural models can be utilised by service designers and users to guide service design and ease the comprehension of services.

Fig. 3. Overview of the approach.

4.2 Business Entities and Relations

Definition 1 (Operation and Parameter). Let $s$ be a service, $OP_s$ is a set of operations of $s$. For each operation $op \in OP_s$, $N(op)$ is the name of $op$, $I(op)$ is the set of input parameters and $O(op)$ is the set of output parameters of $op$.

Let $P$ be a set of parameters. For each $p \in P$, $N(p)$ is the name of $p$, $\gamma(p) \in \{\text{primitive}, \text{complex}\}$ indicates whether $p$ is of a primitive or a complex type (i.e., an user-defined type), and $\text{type}(p)$ specifies the type of data (e.g. string, LineItem) carried by $p$.

$PC = \{p \in P | \gamma(p) = \text{complex}\}$ denotes the set of complex parameters in $P$. $\xi_P \subseteq PC \times P$ specifies the (direct) nesting relation between two parameters. $\lambda^P : \xi_P \rightarrow \{\text{true}, \text{false}\}$ indicates for each $(p, p') \in \xi_P$ whether $p'$ is a compulsory (true) or optional (false) element of $p$.

Definition 2 (Business Entity). $E$ is a set of business entities. For each $e \in E$, $N(e)$ is the name of $e$, $\text{key}(e)$ is the unique identifier of $e$, and $A(e)$ is the set of attributes associated with $e$. For each attribute $a \in A(e)$, $N(a)$ is the name of $a$ and $\text{type}(a)$ is the type of data carried by $a$. In addition, $oprt(e) \in 2^{OP}$ is the set of operations that manipulate $e$, $op^c_e (op^c_e \in oprt(e))$ is the operation that creates an instance of $e$, $op^r_e (op^r_e \in oprt(e))$ is to read an instance of $e$, $op^u_e (op^u_e \in oprt(e))$ is to update an instance of $e$, $op^d_e (op^d_e \in oprt(e))$ is to delete an instance of $e$. 
Definition 3 (Parameter and Business Entity Mapping). Let \( P_C \) be a set of complex parameters, \( \xi^p \) the nesting relation between parameters, and \( E \) a set of business entities. There exists a surjective mapping \( f : P_C \rightarrow E \) where \( \forall p, p' \in P_C, (p, p') \in \xi^p \Rightarrow f(p) \neq f(p') \), i.e. two nesting parameters cannot be mapped to the same business entity.

Definition 4 (Business Entity Nesting Relation). Let \( P_C \) be a set of complex parameters, \( \xi^p \) the nesting relation between parameters, \( E \) a set of business entities, and \( f \) the mapping from \( P_C \) to \( E \). The nesting relation between two business entities can be defined as \( \xi^E \subseteq E \times E \) where \( \forall (e, e') \in \xi^E, \exists p, p' \in P_c \) such that \( f(p) = e, f(p') = e' \), and \( (p, p') \in \xi^p \). This nesting relationship is transitive, i.e., if \( e \xi^E e' \) and \( e' \xi^E e'' \), then \( e \xi^E e'' \).

Definition 5 (Domination, adapted from [4]). Let \( s \) be a service and \( OP_s \) the set of operations of \( s \). Given two business entities \( e, e' \) and two parameters \( p, p' \) s.t. \( e = f(p) \) and \( e' = f(p') \), \( e \) dominates \( e' \) in service \( s \), denoted as \( e \rightarrow e' \), iff: (1) \( \forall op \in OP_s, \) if \( p' \in I(op) \), then \( p \in I(op) \); (2) \( \forall op \in OP_s, \) if \( p' \in O(op) \), then \( p \in O(op) \); (3) \( \exists p \in OP_s, \) s.t. \( p \in I(op) \cup O(op) \), but \( p' \notin I(op) \cup O(op) \).

In other words, \( p \)'s corresponding business entity is \( e \) and \( p' \)'s is \( e' \), \( e \) dominates \( e' \), if and only if (1) for every operation that uses \( p' \) as an input parameter, \( p \) is also used as an input parameter, (2) for every operation that uses \( p' \) as an output parameter, \( p \) is also used as an output parameter, and (3) \( p \) is used by at least one operation (as its input or output parameter) that does not use \( p' \).

Definition 6 (Exclusive Containment). Given two business entities \( e, e' \), \( e \) is exclusively contained in \( e' \) iff \( e \xi^E e' \) and \( e \rightarrow e' \), and \( \neg \exists e'' \in E \setminus \{e\} \) s.t. \( e'' \rightarrow e' \). \( \omega \) captures the set of pairs that represent exclusive containment between two business entities.

Definition 7 (Inclusive Containment). Given two business entities \( e, e' \), \( e' \) is inclusively contained in \( e \) iff \( e \xi^E e' \) and \( e' \rightarrow e \). \( \varphi \) captures the set of pairs that represent inclusive containment between two business entities. \( \lambda^E : \varphi^E \rightarrow \{true, false\} \) indicates for each pair \( (e, e') \in \varphi \) whether \( e' \) is a compulsory or optional element of \( e \). If \( \lambda^E(e, e') = true \), it is called strong inclusive containment, otherwise if \( \lambda^E(e, e') = false \), it is weak inclusive containment. \( \lambda^E(e, e') = \lambda^p(p, p') \) if \( f(p) = e \) and \( f(p') = e' \).

Definition 8 (Association). Given two business entities \( e, e' \), \( e' \) is associated with \( e \) if there exists an operation \( op \) \( (op \in OP_s) \) such that \( e' \) is the primary entity involved in \( op \) and \( key(e) \in I(op) \). \( op \) is called Association Operation. An association operation for \( e \) and \( e' \) is denoted as \( op_{e \leftrightarrow e'}^\psi \). \( \psi \) captures the set of pairs that represent association between two business entities.

Definition 9 (Business Entity Data Model). A business entity based data model (data model for short) \( M \) is a tuple \( (E, \xi^E, \omega, \varphi, \psi) \) which consists of a set business entities \( E \) and their nesting relations \( \xi^E \), exclusive containment relations \( \omega \), inclusive containment relations \( \varphi \), and association relations \( \psi \).
Definition 10 (Business Entity Behavioural Model). A business entity based behavioural model (behavioural model for short) \( P \) is a Petri net \((Q, T, F)\). \( T \) is a set of transitions that specify service operations, \( Q \) a set of places that specify the pre- and post-conditions of service operations, and \( F \subseteq (Q \times T \cup T \times Q) \) a set of flow relations that connect a (pre-)condition to an operation or an operation to a (post-)condition.

4.3 Behavioural Interface Derivation Rules

We consider that for every business entity \( e \) there are four generic types of operations: Create, Read, Update, and Delete (CRUD). They are used to create, retrieve, update, and delete an instance of \( e \) respectively, thus changing the states of business entities. Fig. 4 (a) depicts a typical life cycle of a business entity capturing the state transitions upon carrying out these four types of operations.

The different relations between business entities determine the order of performing the possible types of operations to a business entity. In addition to the CRUD operations (which always apply to one business entity, kind of unary operations), we also consider a so-called Association operation (which apply to two business entities, kind of binary operation). The following rules specify how to derive the occurring order of operations from the business entity relations.

Rule 1 An instance of business entity \( e \) can only be read, updated, or deleted if the instance is created, meaning there is a temporal sequence between \( e \)'s Create operation and its Update, Read, and Delete operations as shown in Fig. 4 (a). Retrieval and deletion of an entity are also permitted when it is in the state of “Updated”.

Rule 2 If business entity \( e' \) is exclusively contained in business entity \( e \), an instance of \( e' \) cannot be created unless an \( e \) is instantiated. In Fig. 4 (b), \( P_c^e \) and \( P_c^{e'} \) represent the behavioural models for creating an instance of \( e \) and \( e' \) respectively. According to the rule, \( P_c^{e'} \) takes place after \( P_c^e \). For example, in FedEx open shipping service, PackageLineItem is exclusively contained in OpenShipOrder and this means an OpenShipOrder has to be created before its PackageLineItem is instantiated.

Rule 3 When a business entity \( e' \) is strongly contained in \( e \) (i.e., \( \lambda^E(e,e') = \text{true} \)), an instance of \( e' \) is required when creating an instance of \( e \). This instance can be either created or read if it exists thereby the instance of \( e' \) can be supplied as part of the input parameters when instantiating \( e \). This rule is depicted by Fig. 4 (c), where \( P_c^{e'} \) represents the behavioural model for creating an instance of \( e' \), and \( t'_r \) denotes transition for retrieving an instance of \( e' \). For example, in FedEx open shipping service, Shipper is strongly contained in OpenShipOrder, so a shipper has to be created or read so that the creation of OpenShipOrder can be carried out. When a business entity \( e' \) is weakly contained in \( e \), there is no specific order between \( e' \)'s and \( e' \)'s creation and it is not compulsory to create an instance of \( e' \) when instantiating \( e \) (Fig. 4 (d)).

Rule 4 When a business entity \( e' \) is associated with \( e \), to form such relationship behaviourally, it is required to attach an instance of \( e' \) to \( e \) after the instance of \( e \) is created. The attachment is achieved by invoking the association
operation $t^{asso}_{e'e}$, but the formation of this association is not a compulsory step of creating an instance of $e$ (Fig. 4 (e)). For example, in Amazon Simple Storage Service (i.e., S3), SetBucketAccessControlPolicy is the association operation that associates a control policy with a bucket, so it is called after a bucket is instantiated to form the association between Bucket and AccessControlPolicy.

Fig. 4. A graphical representation of the rules.

4.4 Service Operation Categorisation

Operations provided by a service can be categorised into five groups: Create, Read, Update, Delete, and Association. Below we define the mapping rules for such categorisation.

**Create** If the invocation of an operation requires some input parameters which are attributes of $e$ and returns a reference to $e$ (i.e., $key(e)$), the operation is for creating an instance of $e$. In other words, an operation that is designed to create an instance of $e$ usually requires its users to pass values of some parameters
which are attributes of $e$. For instance, to create a shipment order, it requires to know details of the shipment order such as shipping date, shipper, and recipient. As a result, the operation should return a reference (e.g., `shipmentNumber`) of the shipment order created.

**Read** If the invocation of an operation requires a value for $key(e)$ and it returns the values of parameters that are attributes of $e$, the operation is for reading an instance of $e$.

**Update** If the invocation of an operation requires values for $key(e)$ and other parameters which are attributes of $e$, the operation is for updating an instance of $e$.

**Delete** If the invocation of an operation requires a value for $key(e)$ and returns nothing related to $e$ but just a status, the operation is for deleting an instance of $e$.

**Association Operation** Given $e'$ is associated with $e$, if the invocation of an operation requires values for some input parameters which are attributes of $e'$ and the reference to another business entity $e$, and it returns a value of the reference to $e'$ (i.e., $key(e')$), the operation attaches an instance of $e'$ to an instance of $e$ and forms the association relationship.

An algorithm, which invokes each operation that manipulates a business entity $e$, analyses the input and output parameters according to the aforementioned rules and then assigns it to $op(c(e))$, $op(r(e))$, $op(u(e))$, or $op(add)$, have been developed, but due to space limit, this paper will not discuss the details. The resulting operations of this algorithm will be utilised in behavioural model derivation algorithms in the following sections.

### 4.5 Behavioural Interface Derivation

Based on the rules in Section 4.3, we can derive behavioural models for an entity’s creation and its life cycle on both abstract and executable (i.e., actual) level. An abstract model is generated in strict compliance with the rules considering only operation types regardless whether an operation can be found. For instance, in FedEx open shipping service, Shipper is strongly contained in OpenshipOrder, meaning a shipper should be instantiated before an openshipOrder is created, but the operation for creating an instance of shipper is not provided by FedEx in reality. A abstract model generates the template according to Rule 3 in Section 4.3 anyway despite the fact that $P_{\text{shipper}}$ is not available. That is to say, an abstract model presents an impeccable behavioural interface for a service, meaning it defines a template, which depicts the ordering constraints that a service should follow. Therefore, this type of model can be utilised as a guidance for service designers when designing services. An executable model, by contrast, considers the availability of an operation and it generates a node only when the corresponding operation can be found. Therefore, the creation of shipper is skipped in generating the executable model for OpenshipOrder’s creation. In other words, an executable model can be utilised by service users to comprehend how to invoke the operations provided by a service. The behavioural
derivation mechanism supports the generation of both abstract and executable models, but this paper focuses on the latter only.

Given a business entity $e$ and the data model of the service that $e$ resides in, we derive a behavioural model $P_e$, which reveals the invocation sequence constraints among the operations provided by the service. Algorithm 1 presents how such a model is generated and Fig. 5 demonstrates the algorithm. Fig. 5 (a) shows an $E_1$ focused data model and Fig. 5 (b) presents the corresponding behavioural model generated by the algorithm.

Specifically, Algorithm 1 consists of five main steps. The first (from line 6 to line 13) involves iterating every business entity $e'$ that is strongly inclusively contained in $e$ and constructing a behavioural model $P_{e'}$ for $e'$. According to Rule 3 in Section 4.3, an instance of $e'$ should be either created or read before creating an instance of $e$. That is to say, the first step of Algorithm 1 is to construct a behavioural model for each $e'$ with $P_{e'}$ and
Algorithm 1 GENERATECREATEBEMODEL

Input: a business entity data model \((E, \xi^E, \omega, \phi, \psi)\), a business entity \(e\) (where \(e \in E\))

1: /* Initialise the business entity behavioural model \(P^e_c\) */
2: \(P^e_c := (Q_e, T_e, F_e)\)
3: \(Q_e := \{q^c_0\} / * q^c_0 is the input (start) place of \(P^e_c\) */
4: \(T_e := \{\tau^c_0\} / * \tau^c_0 is the first silent transition in \(P^e_c\) */
5: \(F_e := \{(q^c_0, \tau^c_0)\}\)
6: /* First step - process strong inclusive containment */
7: for each \(e' \in \varphi(e, e') \land \lambda^E(e, e') = \text{true} \) do
8: \(P'_e := \text{GENERATECREATEBEMODEL}(E, \xi^E, \omega, \phi, \psi, e')\)
9: \(P^e_c := (Q_e \cup Q'_e, T_e \cup T'_e, F_e \cup F'_e)\)
10: \(t^c_0 := \text{CONVERTTOTRANSITION}(\text{op}^c(e'))\)
11: \(T_e := T_e \cup \{t^c_0\} \cup \{\tau^c_0\}\)
12: \(F_e := F_e \cup \{(q^c_0, t^c_0), (t^c_0, q^c_0'), \tau_e, q^c_0', \tau^c_0\}\)
13: end for
14: /* Second step - process the creation */
15: \(Q_e := Q_e \cup \{q^c_{pre}\} \cup \{q^c_{post}\}\)
16: if \(\{\tau^c_0\} \in T\) then
17: \(F_e := F_e \cup \{(\tau^c_0, q^c_{pre}\}\}\)
18: else
19: \(F_e := F_e \cup \{(\tau^c_0, q^c_{pre}\}\}\)
20: end if
21: \(t^c_0 := \text{CONVERTTOTRANSITION}(\text{op}^c(e))\)
22: if \(t^c_0 = \bot\) then
23: return nil
24: end if
25: \(T_e := T_e \cup \{t^c_0\} \cup \{\tau^c_0\}\)
26: \(F_e := F_e \cup \{(q^c_{pre}, t^c_0), (t^c_0, q^c_{post})\}\)
27: /* Third step - process exclusive containment */
28: if \(\omega(e, e') \neq \emptyset\) then
29: for each \(e' \in \omega(e, e')\) do
30: \(P^e_{e'} := \text{GENERATECREATEBEMODEL}(E, \xi^E, \omega, \phi, \psi, e')\)
31: \(\text{REMOVETHEFIRSTPLACE}(P^e_{e'})\)
32: \(\text{REMOVETHEFIRSTFLOW}(P^e_{e'})\)
33: \(P^e_c := (Q_e \cup Q'_{e'}, T_e \cup T'_{e'}, F_e \cup F'_{e'})\)
34: \(F_e := F_e \cup \{(q^c_{post}, \tau_{e'}), (q^c_{e'}, \tau^c_{e'})\}\)
35: end for
36: else
37: \(F_e := F_e \cup \{(q^c_{post}, \tau^c_{e'})\}\)
38: end if
39: /* Fourth step - process weak inclusive containment */
40: for each \(e' \in \varphi(e, e') \land \lambda^E(e, e') = \text{false} \) do
41: \(P^e_{e'} := \text{GENERATECREATEBEMODEL}(E, \xi^E, \omega, \phi, \psi, e')\)
42: \(P^e_c := (Q_e \cup Q'_{e'}, T_e \cup T'_{e'}, F_e \cup F'_{e'})\)
43: \(t_{em} := \text{CREATANEMPTYTRANSITION}()\)
44: \(T_e := T_e \cup \{t_{em}\}\)
45: \(F_e := F_e \cup \{(q^c_{e'}, t_{em}), (t_{em}, q^c_{e'}), (q^c_{e'}, \tau_{e'}), (q^c_{e'}, \tau_{e'})\}\)
46: end for
47: \(Q_e := Q_e \cup \{q^c_{e'}\} / * q^c_{e'} is the output (end) place of \(P^e_{e'}\) */
48: \(F_e := F_e \cup \{(\tau_{e'}, q^c_{e'})\} / * \tau_{e'} is the output transition */
49: /* Fifth step - process association */
50: for each \(e' \in \psi(e, e')\) do
51: \(t^c_{asso} := \text{CONVERTTOTRANSITION}(\text{op}^c_{asso}(ee'))\)
52: \(t_{em} := \text{CREATANEMPTYTRANSITION}()\)
53: \(T_e := T_e \cup \{t^c_{asso}\} \cup \{t_{em}\} \cup \{\tau_{asso}\}\)
54: \(Q_e := Q_e \cup \{q^c_{pre}\} \cup \{q^c_{post}\}\)
55: \(F_e := F_e \cup \{(q^c_{asso}, \tau_{asso}), (\tau_{asso}, q^c_{pre}), (q^c_{pre}, t_{asso})\}\)
56: \(F_e := F_e \cup \{(t^c_{asso}, q^c_{post}, t_{em}, q^c_{post})\}\)
57: end for
58: end
Fig. 5. An abstract demonstration for Algorithm 1.
generates executable models, it exits if the instance of generated before inclusively contained in Petri net model with a transition \( t \). At the end of the first step, the generated \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly as shown in Fig. 4 (c). As each \( e \), the algorithm is recursive, so \( P_e \) may consist of a number of Petri net models. At the end of the first step, the generated \( P_e \) is connected and merged with \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are \( P_e \) before moving on to the next step. In Fig. 5 (a), as no entities are strongly inclusively contained in \( E_1 \), the first step is skipped for \( E_1 \), but \( E_4 \) is strongly inclusively contained in \( E_2 \), so the corresponding nodes (e.g., \( t_{e4}^c \) and \( t_{c4} \)) are
end of the second step, the generated net is connected to $P_c^e$ as its second part. In Fig. 5, this net consists of $q_{pre}^{e_1}$, $t_{c_1}^e$, and $q_{post}^{e_1}$. The third step (from line 27 to line 38) iterates each $e'$ that is exclusively contained in $e$ and generates a behavioural model $P_c^{e'}$ for $e'$. Similarly, the creation of $P_c^{e'}$ is a recursive process and each $P_c^{e'}$ is merged into $P_c^e$ in the end according to Rule 2 in Section 4.3, which is $P_c^e$ can only be called after $P_c^{e'}$. As the first place $q_{pre}^{e'}$ and the first arc $(q_{pre}^{e'}, \tau_0^e)$ are redundant, they are removed before the $P_c^{e'}$ is merged with $P_c^e$. In Fig. 5, the third step processes $E_2$, as it is exclusively contained in $E_1$. As $E_5$ is weakly inclusively contained and $E_4$ is strongly inclusively contained in $E_2$, the algorithm recursively processes these two entities and the output is the Petri net model $P_c^{E_2}$ (as the dotted rectangle indicates in Fig. 5). In step four (from line 40 to line 48), each $e'$ that is weakly inclusively contained in $e$ is checked and a $P_c^{e'}$ is generated. According to Rule 3 in Section 4.3, $P_c^{e'}$ can be skipped, so an empty transition (i.e., $t_{cr}$) is created and incorporated into the $P_c^e$. Similarly to step 1 and step 3, $P_c^{e'}$ is merged into and connected to $P_c^e$ as its third part. In Fig. 5, as $E_5$ is weakly inclusively contained in $E_2$, the corresponding behavioural model $P_c^{E_5}$ is generated and linked to $P_c^{E_2}$. Finally (from line 49 to line 57), entities that are associated with $e$ are iterated and their nets are generated. This step converts the association operation that attaches $e'$ and $e$ to a transition $t_{asso}$ and links it to the end of $P_c^e$. In Fig. 5, as $E_3$ is associated with $E_1$, the corresponding nodes (e.g., $\tau_{asso}$ and $t_{asso}$) are generated and connected to $q_{end}$.

4.6 Deriving Business Entity Life Cycle

Having categorised an entity’s CRUD (i.e., $op^r(e)$, $op^u(e)$, $op^d(e)$), based on Rule 1 in Section 4.3, The life cycle model for an entity can be derived. Algorithm 2 presents a four-step approach, depicting how such a model is generated. The first retrieves the behavioural model for entity $e$’s creation by invoking Algorithm 1 and merges the resulting model into $P_{cycle}$ as the first part of $e$’s life cycle model, as an instance of $e$ should be created before reading, updating, and deleting it. If the behavioural model for entity $e$’s creation is not formed, the whole process terminates. A new silent transition (i.e., $\tau_{cr}^e$) links to $P_{cycle}^e$’s end place $q_{pre}^e$ and it will be connected with $e$’s update, read, and deletion nodes in the following steps. The second step processes $e$’s update. Specifically, it retrieves $op^u(e)$, converts it to the corresponding transition, and then connects it to its pre and post conditions (i.e., places). Another silent transition $\tau_{up}^e$ is introduced in this step and it will be connected to $e$’s read and deleting nodes in the following steps. The third and the fourth steps deal with the transitions that read and delete an instance of $e$. Corresponding nodes are generated and they are linked to $\tau_{cr}^e$ and $\tau_{up}^e$ according to Rule 1 in Section 4.3.

5 Implementation and Validation

To validate the service behavioural interface derivation mechanism, we have developed a Java based prototype, Service Integration Accelerator, which implements the algorithms presented in the previous section and outputs behavioural
models in Petri net graphs by utilising Graphviz\(^2\) and the standard PNML format using a Java based PNML Framework\(^3\). This section presents the details of the experiments we conducted and evaluates the mechanism using their results. All experiments were performed on a laptop with Intel Core i7-3520M CPU 2.90 GHz 4 and 8 GB of memory, running on Ubuntu 14.04 LTS and OpenJDK 1.7 (with standard allocation of memory).

**Hypotheses** Three hypotheses are defined to assess the effectiveness of the mechanism. The first is competence - we presume it can produce abstract behavioural models for every business entity according to the rules in Section 4.3 and executable behavioural models based on the operations provided by a service. Another criterion to be examined is performance - the time taken to derive behavioural models for each business entity should be within one second.

**Objects** Eleven popular services (shown in Table 1) drawn from xmethods.net\(^4\), Amazon.com, and FedEx were chosen as the experiment objects. These samples are from three categories: Internet Services (IS), i.e., services from the Internet, Software-as-a-Service (SaaS), and Enterprise Services (ES) and the complexity of these services increases from IS to ES. Services in the IS category are highlighted in light grey (i.e., the first two services); Services in the SaaS category are dark gray (i.e., the three Amazon services); Services in the ES category are in dim gray (i.e., the six FedEx services).

**Validation Process** We applied the Service Integration Accelerator to the interfaces of the aforementioned 11 services, which cover 115 operations and 621 business entities. Based on the business entity data models generated \(^5\), we run the behavioural interface derivation mechanism to produce the results and then analyse them to assess if they support the hypotheses.

**Results** Table 1 presents the detailed statistics of the generated behavioural models for the 11 services. Specifically, it reports the following details: (1) the number of operations each service provides, (2) the number of business entities, executable behavioural models for entity creation and life cycle generated, (3) The time taken (in milliseconds) for generating these models (with and without PNML output) for each service. The behavioural models for entity creation and life cycle are detailed with number of places, transitions, and flows (i.e., P/T/F in Table 1).

According to the results, Internet services usually do not involve ordering constraints, because they often have only a few operations with a handful of

\(^{2}\) http://www.graphviz.org/
\(^{3}\) http://pnml.lip6.fr/
\(^{4}\) http://www.xmethods.net:5868/ve2/index.po
\(^{5}\) http://www.findpeoplefree.co.uk/findpeoplefree.asmx?wsdl
\(^{6}\) http://ws2.fraudlabs.com/mailboxvalidator.asmx?wsdl
\(^{7}\) http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.wsdl
\(^{8}\) http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
\(^{9}\) http://mechanicalturk.amazonaws.com/AWSMechanicalTurk/2013-11-15/AWSMechanicalTurkRequester.wsdl
\(^{10}\) http://www.fedex.com/us/web-services/
<table>
<thead>
<tr>
<th>Services</th>
<th>Operations</th>
<th>Entities</th>
<th>Executable Models/P/T/F</th>
<th>life cycle models/P/T/F</th>
<th>Elapsed time</th>
<th>Elapsed time with PNML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find People</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MailBox Validator</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Amazon S3</td>
<td>16</td>
<td>17</td>
<td>3/12/9/18</td>
<td>3/22/17/36</td>
<td>166</td>
<td>38051</td>
</tr>
<tr>
<td>Amazon Advertising</td>
<td>9</td>
<td>19</td>
<td>2/12/9/19</td>
<td>2/20/16/36</td>
<td>105</td>
<td>25382</td>
</tr>
<tr>
<td>Amazon Mechanical</td>
<td>44</td>
<td>97</td>
<td>9/36/27/54</td>
<td>9/60/48/104</td>
<td>552</td>
<td>115595</td>
</tr>
<tr>
<td>FedEx Ship</td>
<td>5</td>
<td>168</td>
<td>2/8/6/12</td>
<td>2/12/10/20</td>
<td>165</td>
<td>26210</td>
</tr>
<tr>
<td>FedEx Pickup</td>
<td>3</td>
<td>76</td>
<td>1/4/3/6</td>
<td>1/8/6/13</td>
<td>68</td>
<td>12758</td>
</tr>
<tr>
<td>FedEx Return</td>
<td>1</td>
<td>4</td>
<td>1/4/3/6</td>
<td>1/4/3/6</td>
<td>61</td>
<td>12635</td>
</tr>
<tr>
<td>FedEx Close</td>
<td>6</td>
<td>22</td>
<td>4/20/17/36</td>
<td>4/20/16/34</td>
<td>206</td>
<td>51254</td>
</tr>
<tr>
<td>Open Shipping</td>
<td>22</td>
<td>208</td>
<td>4/20/15/31</td>
<td>4/40/32/74</td>
<td>295</td>
<td>52305</td>
</tr>
<tr>
<td>Address Validation</td>
<td>1</td>
<td>6</td>
<td>1/4/3/6</td>
<td>1/4/3/6</td>
<td>48</td>
<td>12815</td>
</tr>
</tbody>
</table>
parameters and these operations are loosely coupled. For example, the Find People service has only two operations: “findAddress(city, backlinkWebsite)” and “findPeople(exactAddress, backlinkWebsite)”. No business entities have been identified based on these operations, and they can be invoked independently of one another. Therefore, Internet service users will not benefit significantly from the behavioural model derivation mechanism.

As for services in the SaaS category, their interfaces present intermediate complexity. The number of operations provided in the three Amazon web services ranges from 9 to 44. Based on the data model generated and the operations provided by these services, we derived 3, 2 and 9 executable behavioural models for the creation of business entities involved in Amazon S3, Advertising, Mechanical services respectively, and the same number of life cycle models for these entities. Taking S3 as an example, Fig. 6 (a) presents a business entity data model with a focus on Bucket. The generated executable behavioural model for Bucket’s creation is shown in Fig. 6 (b). In this model, the transition: “CreateBucket” has been identified as the one that creates an instance of Bucket. As it can be seen, BucketLoggingStatus is exclusively contained in Bucket, meaning an instance of this entity has to be instantiated after creating an instance of Bucket. “SetBucketLoggingStatus” has been identified as the transition that creates an instance of BucketLoggingStatus, so this operation is called after “CreateBucket” as shown in Fig. 6 (b). In addition, both AccessControlPolicy and Object are associated with Bucket, meaning the attachment of these two entities to Bucket can only take place after an instance of Bucket is created. “SetBucketAccessControlPolicy” and “PutObject” have been identified as the associate operations for AccessControlPolicy and Object respectively, so they can be invoked after $P^C_{bucket}$ to form the association.
Services in the ES category are the most complex ones and they usually involve numerous business entities and operations, so it is significant to derive behavioural models for them. The statistics for the six FedEx services in Table 1 show the number of behavioural models generated. For instance, by analysing the 22 operations provided by FedEx Open Shipping service, we derived 4 executable behavioural models for the creation of the business entities involved in the service. Due to space limit, this paper only presents the one for OpenshipOrder. Fig. 7 (a) depicts a fraction of the OpenshipOrder focused data model and its life cycle model is presented in Fig. 7 (b). As PackageLineItem is exclusively contained in OpenshipOrder, its creation (“addPackagesToOpenShipment”) occurs after OpenshipOrder’s (“createOpenShipment”). There are also other entities such as Shipper, ShippingChargesPayment, Label, and SpecialService that have either exclusive containment or strongly/weak inclusive containment relationships with OpenshipOrder, but no corresponding executable nodes were generated due to the fact that no operations are provided for the creation of these business entities. However, all these nodes were reflected in the abstract behavioural models derived. As the FedEx Open Shipping service provides operations for creating, reading, updating, deleting the core business entities involved (i.e., OpenshipOrder and PackageLineItem), the mechanism was able to categorise them correctly and generate the life cycle model for them (as shown in
Fig. 7 (b)) according to Rule 1 in Section 4.3. The screen shot of the life cycle model in the Service Integration Accelerator environment is also presented in Fig. 8.

![Service Integration Accelerator](image)

Fig. 8. The screen shot of FedEx OpenShipOrder life cycle model.

The time taken to generate these models are listed in the last two columns in Table 1. The elapsed time meets the performance requirement, which is within one second per entity, but producing pnml files for behavioural models takes a large amount of time, with almost 2 minutes for the Amazon Mechanical service at worst. This is because the external PNML library involves intensive IO operations. As the output of pnml is an optional setting in the mechanism, the performance of the core part of the mechanism is not compromised.

**Discussion** The experiments have demonstrated the effectiveness of the mechanism, but we have found several issues at the same time. The first is that the mechanism misses some operations occasionally, especially when it comes to the categorisation of operations. For example, both “PutObjectInline” and “PutObjectInline” should be identified as the association operations, only the later was rendered due to the assumption we made that only one association operation exists. Similarly, in FedEx open shipping service, the operation “confirmOpenShipment” is to confirm the creation of an OpenShipOrder, so it should be invoked at the end of OpenShipOrder creation. However, this operation was missed in the models generated. To address the problem, operations for one category should not be limited to one and a set should be used to keep all operations that fall in the same category. These operations should also be reflected in the behavioural models. Another problem is that the current categorisation algorithm only invokes operations with the minimum set of parameters, and this can sometimes cause inaccuracy and incompleteness. To counter this problem,
more invocations with other alternative sets of parameters should be tried and the responses should then be analysed.

6 Supporting the State-based Patterns

This section demonstrates how the derived behavioural interface models can be utilised to support the patterns depicted in Section 3. Specifically, we drew three services, namely FedEx Shipping, UPS Shipment\(^{11}\) and Xero Invoice service\(^{12}\), and then elaborate service interaction flexible selection and interleaving using their behavioural models.

The service behavioural interface derivation mechanism generates the interfaces for the creation of FedEx, UPS ShipOrder and Xero Invoice as shown in Fig. 9, Fig. 10, and Fig. 11 respectively. Fig. 9 (a) depicts a fraction of the FedEx ShipOrder focused data model and its creation model is presented in Fig. 9 (b). As PackageLineItem is exclusively contained in OpenshipOrder, its creation (“addPackagesToOpenShipment”) occurs after OpenshipOrder’s (“createOpenShipment”). There are also other entities such as Shipper, ShippingCharges-Payment, Label, and SpecialService that have either exclusive containment or strongly/weak inclusive containment relationships with ShipOrder, but no corresponding executable nodes were generated due to the fact that no operations are provided for the creation of these business entities. Similarly, Fig. 10 (a) and Fig. 11 (a) present a sub-version of UPS ShipOrder’s Xero’s Invoice data model (a) and their executable creation behavioural models are shown in Fig. 10 (b) and Fig. 11 (b).

To support Pattern 1 in Section 3, we assume that a shipper would like to ship some goods. The underlying task is to create a shipment order, so in this shipment process the current state is “Initial” \((q_c\) in Fig. 12) and the desired one is “ShipOrder created” \((q_d\) in Fig. 12). As the behavioural models for creating a ShipOrder of FedEx and UPS have been derived, the shipper can flexibly elect an interaction set to complete the state transition. An interaction set consists a number of externally exposed service operations, which are generated by the service behavioural interface derivation mechanism. As shown in Fig. 12, there are two transitions \((P_1 \text{ and } P_2)\) which can progress the state from “Initial” to “ShipOrder created” and they are mapped to two interactions with FedEx Shipping and UPS Shipping as shown in the sequence diagram in Fig. 12.

To support Pattern 2 in Section 3, we assume that, in a procurement process, a purchasing party acts as a shipper that arranges the shipment for its customers. The purchasing service interacts with FedEx shipping service to create a ShipOrder and this interaction involves two steps according to the behavioural models generated (in Fig 9): invoking “createOpenShipment” and “addPackagesToOpenShipment”. The purchasing party also desires to interface with the invoicing service - Xero invoicing in order to issue an invoice to its customers and the creation of such an invoice consists of three invocations as the behavioural

\(^{11}\) https://www.ups.com/upsdeveloperkit?loc=en_US

\(^{12}\) https://developer.xero.com/documentation/api/invoices/
models in Fig. 11 indicate. As the Invoice and the ShipOrder do not have dependencies, these two interactions can be run in parallel. The essence of Pattern 2 is that interactions can be interleaved based on states of a service conversation, and this can be achieved by using the state of business entities given that the service behavioural interface derivation mechanism produces these states and the corresponding protocols that can realise their transitions. Based on the behavioural models in Fig. 9 and 11, we can derive the petri net based behavioural model that depicts two sets of interleaved interactions as presented in Fig 13 (a). The state transition from \( q_3 \) to \( q_4 \) implements the creation of FedEx ShipOrder and the one from \( q_1 \) to \( q_2 \) realises the creation of Xero invoice. This two transitions
are mapped to two interleaved interactions as shown in Fig 13 (b). To conclude, service interface interleaving can be enabled and supported through the implementation of state-based service protocols generated by the behavioural interface derivation mechanism. However, one of the open issues is that we currently do not support fine-grained states within the cycle of creating, reading, updating and deleting an business entities and this will be addressed in our future work.

With the state-based behavioural model, we also can support another two patterns: “Cancel interactions to previous state” and “Replace interactions to future state”. The former refers to interactions with one or more services are cancelled to a previous state in a conversation. This is achieved by issuing can-
cellations for all interactions that are currently taking place or have completed between the current state of a conversation and a previous state. Several sets of interactions may be cancelled across several state transitions, as a result. The latter involves cancelling interactions to a previous state in a conversation, and executing interactions to a future state (e.g. the current state as the point of cancellation), allowing parts of a conversation with existing services to be replaced. Due to space limit, the support of these two patterns will be discussed in the future work.

7 Conclusion

This paper presented a service behavioural interface derivation mechanism, which generates service behavioural interfaces based on the core artefact reflected in services - business entities and the relationships between them. We validated the mechanism using a variety of services ranging from internet services to enterprise services. The study has demonstrated that the business entity based interface derivation technique is an effective solution to deriving behavioural models for entity’s creation and life cycle both on abstract and executable level. The resulting models of the mechanism can be utilised in a service integration scenario, where behavioural interfaces of services are unknown, and it can also provide a guidance to service designers. The states provide a declarative approach for conversation needs, without prescribing which services or which order of interactions should be taken. This opens up the possibility of dynamically determined execution of conversation, like the services that are relevant to advancing states, the
interactions involved in fulfilling conversation progress and interleaving interac-
tions across different services beyond established protocols of individual services.
Advanced operations like cancellations back to previous states and replacements
with new providers going forward, also become possible. To the best of the
authors’ knowledge, such a declarative approach is the first pitch to support
flexible leverage of wide spanning services in global business networks. Future
work includes the improvement of the operation categorisation and analysis of
the models generated. As for the former, we will allow more than one operations
to fall in one category and adopt a Monta Carlo statistic approach to search
for other valid service invocations so that a complete input and output param-
eter analysis can be carried out. For the later, Petri net deadlock detection and
reachability analysis techniques will be utilised to optimise the models.

References

1. Mateescu, R., Poizat, P., Salaun, G.: Adaptation of service protocols using pro-
cess algebra and on-the-fly reduction techniques. IEEE Transactions on Software
2. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-
automated adaptation of service interactions. In: Proceedings of the 16th inter-
national conference on World Wide Web (WWW’07), New York, NY, USA, ACM
(2007) 993–1002
behavior protocols for composable web-services. In: Proceedings of the the 7th
joint meeting of the European software engineering conference. ESEC/FSE ’09,
New York, NY, USA, ACM (2009) 141–150
centric models of business processes. In Bellahsne, Z., Lonard, M., eds.: Advanced
Information Systems Engineering. Volume 5074 of Lecture Notes in Computer
web services. In: Advanced Information Systems Engineering (27th International
Conference on Advanced Information Systems Engineering). Lecture Notes in Com-
automata learning. In Margaria, T., Steffen, B., eds.: Leveraging Applications of
Formal Methods, Verification, and Validation. Volume 6416 of Lecture Notes in
from imperfect service interaction logs. In: IEEE 23rd International Conference on
9. Ragab Hassen, R., Nourine, L., Toumani, F.: Protocol-based web service com-
position. In: Proceedings of the 6th International Conference on Service-Oriented
Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Vol-