This is the author's version of a work that was submitted/accepted for publication in the following source:

Discovering behavioural interfaces for overloaded web services. In 2015 IEEE World Congress of Services (SERVICES), IEEE, New York City, NY, pp. 286-293.

This file was downloaded from: http://eprints.qut.edu.au/91739/

© Copyright 2015 IEEE

Notice: Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:

http://doi.org/10.1109/SERVICES.2015.50
Discovering Behavioural Interfaces for Overloaded Web Services

Fuguo Wei, Chun Ouyang, Alistair Barros
Information Systems School
Queensland University of Technology
Brisbane, Australia
Email: {f.wei, c.ouyang, alistair.barros}@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 to improving the efficiency of integration with these services. Interfaces of enterprise systems are typically larger, more complex and overloaded, with single operation 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 behavioural interface derivation.

I. INTRODUCTION

Conventional service adaptation [1] 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 [2]. 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. For example, FedEx Web services have more than 1000 parameters in some of their operations, while SAP enterprise services have up to 400 parameters. The availability of behavioural interfaces is not guaranteed in practice [3]. Even if it is, behavioural interfaces, as structural ones, present ambiguities because of their overloading nature, resulting in different variants that may lead to differences in service interactions, and all of which are optional and determined through run-time as to which choice of interactions is required. Therefore, service users cannot easily determine which particular part of the behavioural protocol applies for interacting with a service.

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 [4], [5]. 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 [6]. Through [7], 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. This paper extends 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. II 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. III. Sect. IV evaluates the...
mechanism by experimenting the implemented prototype with a variety of services and reveals some open issues. Finally, Sect. V concludes the paper and outlines the future work.

II. 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. [8] 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. [9] 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ootnote{http://www.w3.org/Submission/WSDL-S/}, 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 [3] 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” [10]. However, this technique assumes that the behavioural interfaces of individual services involved in a composition are available.

Another proposal by Bertolino et al. [4] 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. [5] 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 information entities called domination, which has been adapted in our study to support the derivation of relationships between business entities.

III. BEHAVIOURAL INTERFACE DERIVATION

A. Overview of the Approach

We propose a four-step approach for deriving service behavioural interface, which is illustrated in Fig. 1. Given a service specification such as a WSDL file, the first step is to identify business entities and to 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 [7]. 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.

B. Business Entities and Relations

Structural input and output interfaces of operations on a service are mapped to a business entity based service data model. Fig. 2 presents an example of such a model, where four business entities (there could be more in a real example) are mapped from the complex parameters in the input interface of operation 1 (op1) on the service s. For example, p1 is mapped to business entity A, and the parameters under p1 are then mapped as attributes of the business entity. Each business entity has a key, which is identified through one of the nested parameters. For instance, p2 is mapped as the key of A. Because p4 is nested in p1, it implies that its corresponding business entity (i.e., B) has a relationship with p1’s corresponding business entity (i.e., A). Specifically, three types of relationships are derived in this study, namely Exclusive Containment, Inclusive Containment, and Association.

Domination, adapted from [5], if two business entities e and e’ are derived from two parameters p and p’, e dominates e’ if and only if in the context of a service (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’.

Exclusive Containment, e’ is exclusively contained in e iff e dominates e’ and ¬∃ e’” that e” dominates e’. ω captures the
Inclusive Containment, the relationship between \( e \) and \( e' \) is inclusive containment iff \( e' \) dominates \( e \). \( \varphi \) captures the set of pairs that represent inclusive containment between two business entities. If \( e' \) is a compulsory part of \( e \) (this is inferred by the relation between the corresponding parameters), it is called **Strong Inclusive Containment**, otherwise it is a **Weak inclusive containment**.

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

### C. Behavioural Interface Derivation Rules

**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.

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. 3 (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. 3 (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. 3 (b), \( P^{{e'}} \) and \( P^{{e'}}_{\text{c}} \) represent the behavioural models for creating an instance of \( e \) and \( e' \) respectively. According to the rule, \( P^{{e'}}_{\text{c}} \) takes place after \( P^{{e'}} \). 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 the relationship between e and e′ is strong inclusive containment, 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. 3 (c), where P_e′ represents the behavioural model for creating an instance of e′, and t_e′ denotes transition for retrieving an instance of e′. For example, in FedEx open shipping service, the relationship between OpenshipOrder and Shipper is strong inclusive containment, so a shipper has to be created or read so that the creation of OpenshipOrder can be carried out. When the relationship between these two entities is weak inclusive containment, 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. 3 (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 ρ_{asso_e′}, but the formation of this association is not a compulsory step of creating an instance of e (Fig. 3 (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.

---

**D. 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, 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 to categorise operations has been developed, but due to space limit, this paper will not discuss the details. The resulting operations of this algorithm will be utilised in behaviour model derivation algorithms in the following sections.

---

**E. Behavioural Interface Derivation**

Based on the rules in Section III-C, we can derive behaviour 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, the relationship between OpenshipOrder and Shipper is strong inclusive containment, 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 III-C anyway despite the fact that P_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. 4 demonstrates the algorithm. Fig. 4 (a) shows an E1 focused data model and Fig. 4 (b) presents the corresponding behavioural model generated by the algorithm.
Algorithm 1 \textsc{GenerateCreateBEModel}

\textbf{Input:} a business entity data model \((E, \omega, \phi, \psi)\), a business entity \(e\) (where \(e \in E\))

\begin{enumerate}
\item \(/ * \text{Initialise the business entity behavioural model } P^e_c */\)
\item \(P^e_c := (Q_e, T_e, F_e)\)
\item \(Q^e_0 := \{q^e_0\} / * q^e_0 \text{ is the input (start) place of } P^e_c */\)
\item \(T^e_0 := \{\tau^e_0\} / * \tau^e_0 \text{ is the silent first transition in } P^e_c */\)
\item \(F_e := \{\{q^e_0, \tau^e_0\}\}\)
\item \(/ * \text{First step - process strong inclusive containment */}\)
\item \textbf{for} each \(e' \in \varphi(e, e') \wedge \lambda^E(e, e') = \text{true} \textbf{do}\)
\item \(P^{e'}_C := \textsc{GenerateCreateBEModel}(E, \omega, \phi, \psi, e')\)
\item \(P^e_C := (Q_e \cup Q^{e'}, T_e \cup T^{e'}, F_e \cup F^{e'})\)
\item \(t^e_{e'} := \textsc{ConvertToTransition}(op^e(e'))\)
\item \(T_e := T_e \cup \{ t^e_{e'} \} \cup \{ \tau^e_0 \}\)
\item \(F_e := F_e \cup \{ \{q^e_0, \tau^e_0\}, (q^e_0, \tau^e_0) \}\)
\item \textbf{end for}\)
\item \(/ * \text{Second step - process the creation */}\)
\item \(Q_e := Q_e \cup \{q^e_{post}\} \cup \{q^e_{post}\}\)
\item \textbf{if} \(\{\tau^e_0\} \in T \textbf{then}\)
\item \(F_e := F_e \cup \{\{\tau^e_0, q^e_{pre}\}\}\)
\item \textbf{else}\)
\item \(F_e := F_e \cup \{\{\tau^e_0, q^e_{pre}\}\}\)
\item \textbf{end if}\)
\item \(t^e_{e'} := \textsc{ConvertToTransition}(op^e(e'))\)
\item \textbf{if} \(t^e_{e'} = \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\item \(/ * \text{Third step - process exclusive containment */}\)
\item \textbf{for} each \(e' \in \omega(e, e') \neq \emptyset \textbf{do}\)
\item \(P^{e'}_C := \textsc{GenerateCreateBEModel}(E, \omega, \phi, \psi, e')\)
\item \textbf{if} \(\{\tau^e_0\} \in T \textbf{then}\)
\item \(F_e := F_e \cup \{\{\tau^e_0, q^e_{pre}\}\}\)
\item \textbf{else}\)
\item \(F_e := F_e \cup \{\{\tau^e_0, q^e_{pre}\}\}\)
\item \textbf{end if}\)
\item \(t^e_{e'} := \textsc{ConvertToTransition}(op^e(e'))\)
\item \textbf{if} \(t^e_{e'} = \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\item \(/ * \text{Fourth step - process weak inclusive containment */}\)
\item \textbf{for} each \(e' \in \varphi(e, e') \wedge \lambda^E(e, e') = \text{false} \textbf{do}\)
\item \(P^{e'}_C := \textsc{GenerateCreateBEModel}(E, \omega, \phi, \psi, e')\)
\item \(P^e_C := (Q_e \cup Q^{e'}, T_e \cup T^{e'}, F_e \cup F^{e'})\)
\item \textbf{end for}\)
\item \(/ * \text{Create an empty transition */}\)
\item \(t_{\text{em}} := \textsc{CreateANEmptyTransition}()\)
\item \(T_e := T_e \cup \{ t_{\text{em}} \}\)
\item \(F_e := F_e \cup \{\{q^e_{post}, \tau^e_0\}, t_{\text{em}}, q^e_{em}, \tau^e_0\}\)
\item \textbf{end for}\)
\item \(/ * \text{Fourth step - process association */}\)
\item \textbf{for} each \(e' \in \psi^e(e, e') \textbf{do}\)
\item \(t^e_{asso} := \textsc{ConvertToTransition}(op^{asso}(ee'))\)
\item \(t_{\text{em}} := \textsc{CreateANEmptyTransition}()\)
\item \(T_e := T_e \cup \{ t^e_{asso} \} \cup \{ t_{\text{em}} \} \cup \{ \tau_{asso} \}\)
\item \(Q_e := Q_e \cup \{q^e_{em}, \tau_{asso}\}\)
\item \(F_e := F_e \cup \{\{t^e_{asso}, q^e_{asso}\}, t_{\text{em}}, q^e_{em}, \tau_{asso}\}\)
\item \(F_e := F_e \cup \{\{t^e_{asso}, t_{\text{em}}, q^e_{em}, \tau_{asso}\}\}\)
\item \textbf{end for}\)
\item \(/ * \text{Fifth step - process association */}\)
\item \textbf{for} each \(e' \in \psi^e(e, e') \textbf{do}\)
\item \textbf{return} \(P^e_C\)
\end{enumerate}

Legend

\begin{itemize}
\item Exclusive containment
\item Strong inclusive containment
\item Weak inclusive containment
\item Association
\end{itemize}

Fig. 4: An abstract demonstration for Algorithm 1.

Algorithm 2 \textsc{GenerateEntityLifeCycle}

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

\begin{enumerate}
\item \(p_{preC} := (Q, T, F)\)
\item \(/ * \text{Step 1 - the behavioural model for creating */}\)
\item \(P^e_c := \textsc{GenerateCreateBEModel}(E, \xi^E, \omega, \phi, \psi, e)\)
\item \textbf{if} \(P^e_c \neq \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\item \(T^e := T^e \cup \{ t^e_{em} \} = \{ \tau^e_{up} \}\)
\item \(Q := Q \cup \{q^e_{preUp}\} \cup \{q^e_{postUp}\}\)
\item \(F := F \cup \{\{\tau^e_{up}, q^e_{preUp}\}\}\)
\item \textbf{if} \(t^e_{e'} = \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\item \(T^e := T^e \cup \{ t^e_{em} \} = \{ \tau^e_{up} \}\)
\item \(Q := Q \cup \{q^e_{preUp}\} \cup \{q^e_{postUp}\}\)
\item \(F := F \cup \{\{\tau^e_{up}, q^e_{preUp}\}\}\)
\item \textbf{if} \(t^e_{e'} \neq \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\item \(T^e := T^e \cup \{ t^e_{em} \} = \{ \tau^e_{up} \}\)
\item \(Q := Q \cup \{q^e_{preUp}\} \cup \{q^e_{postUp}\}\)
\item \(F := F \cup \{\{\tau^e_{up}, q^e_{preUp}\}\}\)
\item \textbf{if} \(t^e_{e'} = \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\item \(T^e := T^e \cup \{ t^e_{em} \} = \{ \tau^e_{up} \}\)
\item \(Q := Q \cup \{q^e_{preUp}\} \cup \{q^e_{postUp}\}\)
\item \(F := F \cup \{\{\tau^e_{up}, q^e_{preUp}\}\}\)
\item \textbf{if} \(t^e_{e'} = \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\item \(T^e := T^e \cup \{ t^e_{em} \} = \{ \tau^e_{up} \}\)
\item \(Q := Q \cup \{q^e_{preUp}\} \cup \{q^e_{postUp}\}\)
\item \(F := F \cup \{\{\tau^e_{up}, q^e_{preUp}\}\}\)
\item \textbf{if} \(t^e_{e'} \neq \bot\textbf{then}\)
\item \textbf{return} \textit{nil}\)
\item \textbf{end if}\)
\end{enumerate}
Specifically, Algorithm 1 consists of five main steps. The first (from line 6 to line 13) involves iterating every business entity $e'$ that has strong inclusive containment relationship with $e$ and constructing a behavioural model $P_{e'}^{c}$ for $e'$. According to Rule 3 in Section III-C, 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 $P^{c}_{e'}$ and $t^{c}_{e'}$ as shown in Fig. 3 (c). As each $e'$ may further contain other business entities, the algorithm is recursive, so $P^{c}_{e'}$ may consist of a number of Petri net models. At the end of the first step, the generated $P^{c}_{e'}$ is connected and merged with $P^{c}_{e}$ before moving on to the next step. In Fig. 4 (a), as no entities have strong inclusive containment relationship with $E_{1}$, the first step is skipped for $E_{1}$, but $E_{4}$ has strong inclusive containment relationship with $E_{2}$, so the corresponding nodes (e.g., $t^{c}_{e4}$ and $t^{c}_{e4}$) are generated before $t^{c}_{e2}$. The second step (from line 14 to line 26) constructs a Petri net model with a transition $t^{c}_{e}$, which represents the operation that creates an instance of $e$, and its pre-condition and post-condition. As this algorithm generates executable models, it exits if $t^{c}_{e}$ is not found in the service. At the end of the second step, the generated net is connected to $P^{c}_{e}$ as its second part. In Fig. 4 (b), this net consists of $q^{pre}_{c}$, $t^{c}_{e1}$, and $q^{post}_{c}$. 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 III-C, which is that $P^{c}_{e}$ can only be called after $P^{c}_{e'}$. As the first place $q^{c}_{e'}$ and the first arc ($q^{c}_{e'}$, $t^{c}_{e}$) are redundant, they are removed before the $P^{c}_{e}$ is merged with $P^{c}_{e'}$. In Fig. 4 (b), the third step processes $E_{2}$, as it is exclusively contained in $E_{1}$. As the relationship between both $E_{3}$ and $E_{4}$ and $E_{2}$ is weak inclusive containment, the algorithm recursively processes these two entities and the output is the Petri net model $P^{c}_{e2}$ (as the dotted rectangle indicates in Fig. 4 (b)). Step four (from line 40 to line 48), each $e'$ that has weak inclusive containment relationship with $e$ is checked and a $P^{c}_{e'}$ is generated. According to Rule 3 in Section III-C, $P^{c}_{e'}$ can be skipped, so an empty transition (i.e., $t^{c}_{em}$) 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. 4, as the relationship between $E_{5}$ and $E_{2}$ is weak inclusive containment, the corresponding behavioural model $P^{c}_{e5}$ is generated and linked to $P^{c}_{e2}$. 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 $t^{asso}_{c}$ and links it to the end of $P^{c}_{e}$. In Fig. 4, as $E_{3}$ is associated with $E_{1}$, the corresponding nodes (e.g., $t^{asso}_{c}$ and $t^{asso}_{c}$) are generated and connected to $q^{c}_{0}$.

F. Deriving Business Entity Life Cycle

Having categorised an entity’s CRUD, based on Rule 1 in Section III-C. 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 $P^{cycle}_{e}$ 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., $\tau_{up}$) links to $P^{c}_{e}$’s end place $q^{c}_{0}$ 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 $op^{e}(e)$, converts it to the corresponding transition, and then connects it to its pre and post conditions (i.e., places). Another silent transition $\tau_{up}$ 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 $\tau_{up}$ and $\tau_{up}$ according to Rule 1 in Section III-C.

IV. 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 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 III-C 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 I) 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 grey (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 [7], we run the behavioural interface derivation mechanism to produce the results and then analyse them to assess if they support the hypotheses.

Results Table I 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

\(^2\)http://www.graphviz.org/
\(^3\)http://pnml.lip6.fr/
\(^4\)http://www.xmethods.net:5868/ve2/index.po
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 I).

TABLE I: Behavioural interface derivation results.

<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>1</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>3/12/19/18</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>2/12/19/19</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>9/36/22/54</td>
<td>9/04/08/104</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>12358</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>12135</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>6</td>
<td>1</td>
<td>1/4/3/6</td>
<td>1/4/3/6</td>
<td>48</td>
<td>12815</td>
</tr>
</tbody>
</table>

According to the results, Internet services usually do not involve ordering constraints, because they often have only a few operations with a handful of 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. 5 (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. 5 (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 has been identified as the transition that creates an instance of Bucket-LoggingStatus, so this operation is called after “CreateBucket” as shown in Fig. 5 (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. 6 (a) depicts a fraction of the OpenshipOrder focused data model and its life cycle model is presented in Fig. 6 (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. 6 (b)) according to Rule 1 in Section III-C.

The time taken to generate these models are listed in the last two columns in Table I. 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.

V. CONCLUSION AND OUTLOOK

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. 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 Monte Carlo statistic approach to search for other valid service invocations so that a complete input and output parameter analysis can be carried out. For the later, Petri net deadlock detection and reachability analysis techniques will be utilised to optimise the models.

REFERENCES


