Deriving Artefact-Centric Interfaces for Overloaded Web Services

Fuguo Wei(✉), Alistair Barros, and Chun Ouyang

Queensland University of Technology, Brisbane, Australia
`{f.wei,alistair.barros,c.ouyang}@qut.edu.au`

Abstract. We present a novel framework and algorithms for the analysis of Web service interfaces to improve the efficiency of application integration in wide-spanning business networks. Our approach addresses the notorious issue of large and overloaded operational signatures, which are becoming increasingly prevalent on the Internet and being opened up for third-party service aggregation. Extending upon existing techniques used to refactor service interfaces based on derived artefacts of applications, namely business entities, we propose heuristics for deriving relations between business entities, and in turn, deriving permissible orders in which operations are invoked. As a result, service operations are refactored on business entity CRUD which then leads to behavioural models generated, thus supportive of fine-grained and flexible service discovery, composition and interaction. A prototypical implementation and analysis of web services, including those of commercial logistic systems (FedEx), are used to validate the algorithms and open up further insights into service interface synthesis.

Keywords: Web service · Business entity · Service interface synthesis · Service interface analysis

1 Introduction

Services have become the standard way of exposing applications, beyond technology and organisational boundaries, to allow functional capabilities to be accessed and composed, based on loosely coupled collaborations. Web Services Description Language (WSDL), REST and proprietary application programming interfaces (API) are rapidly growing, especially with the rise of Internet-based applications, software-as-a-service and cloud computing, in turn leading to larger vendors providing openly available interfaces for large and otherwise "in-house" software installations, notably in the enterprise systems segment (e.g. ERP and CRM systems). Consequently, the number, range of types, size, and overall complexity of services are progressively increasing, compounding the challenges of

---

This work is sponsored by Smart Service CRC Australia and in part by ARC Discovery Grant DP140103788.

J. Zdravkovic et al. (Eds.): CAiSE 2015, LNCS 9097, pp. 501–516, 2015.
DOI: 10.1007/978-3-319-19069-3_31

integrating services and vastly outpacing the conventional means to adapt services to support application interoperability in diffuse network settings.

Consider, for example, SAP's Enterprise (Web) services, as analysed in [1], where the input message of the goods movement service operation has 104 parameters, out of which only 12 are mandatory for the correct invocation of the service, with the other 92 data types and properties factoring in different usages of the service across different industries. As another example, FedEx provides 12 Web services[1] with the open shipping service demonstrating the highest complexity. It has 22 operations with the average number of input parameters of 307; 111 of them are nested types on average; an average number of hierarchical levels is 6. The operation which has the largest number of input parameters is createOpenShipment, with 1335 parameters, 442 of which are complex and the total levels of hierarchy are 9. More contemporary services from Internet players, too, have non-trivial operations as seen in the e-commerce services provided by Amazon, requiring comprehension of large and technically intricate documentation[2]. From a service consumer's perspective, several problems arise from this level of service complexity: (1) It is difficult to comprehend what each parameter means as the number is very large (2) It is challenging to know which parameter should go along with which for a particular purpose, as a large number of these parameters are optional and there are certain dependencies between them which consumers often do not have sufficient comprehension. For example, parameter B may become compulsory because parameter A is used in a specific invocation (3) How are operations related to each other? In other words, are there any sequence constraints to invoke these operations? (4) How are operations in one service related to ones in other services?

In general, integration of heterogeneous applications requires adapters to mediate across data types of operations (structural aspects) and the permissible orders in which operations are invoked (behavioural aspects). Despite advances in automated support for service adaptation, complex interfaces such as those from SAP, FedEx, Amazon and many others require manual effort and reliance on providers or specialist integrators, to design and implement the required service adapters [2]. In fact, most service providers, especially enterprise systems vendors, do not disclose structural and behavioural interfaces needed for service adaptation, but instead publish interface signatures such as WSDL specifications [3]. Thus, service adaptation incurs significant lead times and costly maintenance to yield service adapters, and their productivity in the context of dynamic service growth on the scale of the Internet is restricted.

This paper extends upon a recent and complementary strategy to conventional service adaptation, whereby the details of service interfaces and knowledge required to interact with them can be unilaterally synthesised by service consumers. Existing interface synthesis techniques build on 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*,

---

[1] http://www.fedex.com/us/web-services/

[2] http://aws.amazon.com/ecommerce-applications/

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. We extend upon these techniques to derive key *relationships* between business entities, which then provides a refined understanding of derived business entities and their dependencies, allowing invocation dependencies across their operations to be derived. For example, if one business entity such as a purchase order strictly contains another business entities such as line items, the container business entity should be created before the contained entities. In all, we develop heuristics for three relationship categories: strong containment, weak containment, 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. Section 2 reviews state of the art and this is followed by the elaboration on the key components of our interface synthesis framework and the development of detailed insights into its most novel features in Section 3. Section 4 evaluates the framework by experimenting the implemented prototype with a variety of services and reveals some open issues. Finally, Section 5 concludes the paper and outlines the future work.

2 Related Work

Service analysis techniques have been proposed over many years to address challenges of service integration concerning structural and behavioural aspects of interfaces. Different approaches have been proposed including the utilisation of semantic ontologies to annotate interfaces to facilitate discovery, use, and composition of services. As an example, Falk et al. [6] adapted automata learning to the problem of service interaction analysis. This proposal usefully combines automated analysis with semantic ontologies in that it needs semantically annotated interface descriptions showing preconditions and effects as the prerequisite to learn interaction protocols. These semantically annotated descriptions are usually not provided by service providers in practice, and the development and maintenance of semantic ontologies requires significant lead times and adoption. Complementary to semantic techniques, log mining algorithms [7] have been proposed for matching data type of target services for service requests, which can also be used at design-time to develop adapters. These incur overheads for aggregating logs and can suffer from missing information for derivation of association dependencies. As we discussed above, our approach concerns static analysis of service interfaces in order to derive the structure and behaviour of services, complementary to semantic and mining approaches.

The first challenge of static service interface analysis is to identify business entities in operations, noting that operations of especially larger systems can have more than one entity, with potential overloading arising from bad service interface design. Identification of business entities is a complex problem requiring an insight into the clustering of different attributes which imply structural type cohesion of an entity. Proposals for static interface analysis proceed from assumptions of attribute to entity type associations based on the use of prior matching techniques (ontology or mining based). The approach of Kumaran et al. [5] proposes heuristics for understanding basic business entity relationships based on the domination theory of business entities, however the derived relationship type is strict containment, which leads to a limited understanding of operation invocation dependencies across services. A more advanced proposal for behavioural interface synthesis has been proposed by [4] based on data dependencies between service operations' input and output parameters, but the study has not been exposed to overloaded service interfaces such as the aforementioned examples from enterprise and Internet players.

Service composition approaches have also been used in the context of service adaptation, the common problem being addressed is "how to automatically generate a new target service protocol by reusing some existing ones" [8]. However, this technique assumes that the interfaces of individual services involved in a composition are available.

3 Service Interface Synthesis

To address the aforementioned problems, this section presents the details of a service interface synthesis framework. It consists of two main modules: Service interface analysis and Service interface refining.

The service interface analysis module is comprised of two components (Fig. 1): *BE data model derivation* and *Service operation refactoring*. They analyse service structural interfaces and determine the order of invoking operations provided by a service. Services essentially focus on addressing and transferring states of resources and business entities are the primary resource manipulated by services in the context of global business networks. Therefore, our service analysis is carried out based on the notion of business entity (i.e., entity types) - a concept widely adopted in Object Role Modelling [9]. The *BE data model derivation* component analyses the input and output parameters of operations on a service and map them to a business entity-based service data model (BE data model). The *Service operation refactoring* component categorises operations provided by a service according to what each operation does to a business entity (i.e, to CREATE, READ, UPDATE or DELETE (CRUD) a business entity). This component also generates behavioural interfaces for each CRUD operation of a business entity.

As a result of structural and behavioural interface analysis, a complex service interface is mapped onto a BE data model and a behavioural interface model. The former presents business entities and the relations among them inherent in the service, and the later depicts the behavioural constraints that service consumers are required to follow. These models form the refactored service interfaces, which encapsulate and simplify complex and overloaded service interfaces. Having these structured service interfaces, valid combinations of input parameter sets can be easily derived. The *Service interface refining* component then utilises a Monte Carlo statistic approach [10] to search for possible valid combinations and then invoke services using these combinations with sample data values in order to determine and test their validity. An Interface Layer is formed in the end, and it exhibits much simpler service interface with possible valid combinations and behavioural specifications so that service consumers can easily consume services. Due to space limit, this paper only addresses the first module: service interface analysis, which simplifies service structural interfaces and generates service behavioural interfaces.

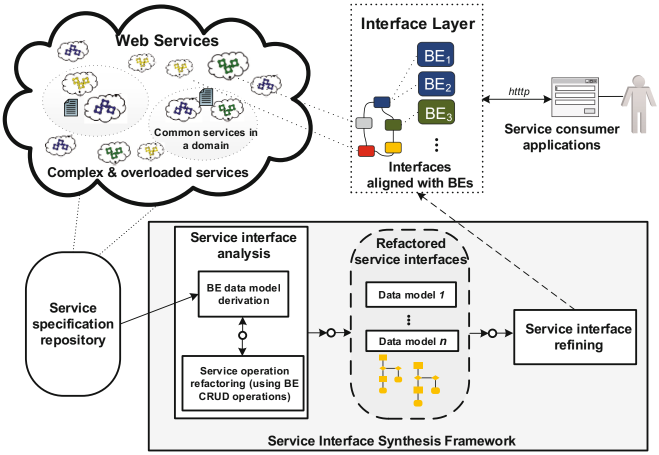
Web Services
Common services in a domain
Complex & overloaded services
Interface Layer
BE_1
BE_2
BE_3
Interfaces aligned with BEs
htttp
Service consumer applications
Service specification repository
Service interface analysis
BE data model derivation
Service operation refactoring (using BE CRUD operations)
Refactored service interfaces
Data model 1
Data model n
Service interface refining
Service Interface Synthesis Framework

Fig. 1. An overview of the service interface synthesis framework

3.1 BE Data Model Derivation

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

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

$P_C = \{p \in P | \gamma(p) = complex\}$ denotes the set of complex parameters in P. $\xi^P \subseteq P_C \times P$ specifies the (direct) nesting relation between two parameters. ξ^P is transitive and irreflexive. $\lambda^P : \xi^P \rightarrow \{true, false\}$ 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 $e \in E$, $\mathcal{N}(e)$ is the name of e, $key(e)$ is the unique identifier of e, and $\mathcal{A}(e)$ is the set of attributes associated with e. For each attribute $a \in \mathcal{A}(e)$, $\mathcal{N}(a)$ is the name of a and $type(a)$ is the type of data carried by a. □

Definition 3 (Parameter and Business Entity Mapping). Let P_C be a set of complex parameters, ξ^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, ξ^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 $e\xi^E e''$. $\lambda^E : \xi^E \rightarrow \{\mathit{true}, \mathit{false}\}$ indicates for each $(e, e') \in \xi^E$ whether e' is a compulsory (*true*) or optional (*false*) element of e. $\lambda^E(e, e') = \lambda^P(p, p')$ if $f(p) = e$ and $f(p') = e'$. □

Definition 5 (Domination, adapted from [5]). 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 $e \mapsto e'$, 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)$\exists\ op \in OP_s$, 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 (Strong Containment). Given two business entities e and e', e' is strongly contained in e iff $e\xi^E e'$, $e \mapsto e'$, and $\nexists e'' \in E \setminus \{e\}$ s.t. $e'' \mapsto e'$. τ captures the set of strong containment relations between business entities. □

Definition 7 (Weak Containment). Given two business entities e and e', e' is weakly contained in e iff $e\xi^E e'$ and $e' \mapsto e$. φ captures the set of weak containment relations between business entities. □

Definition 8 (Association). Given two business entities e and e' where $(e, e') \in \varphi$, e and e' are associated with each other if there exist two parameters p, p' such that $e = f(p)$ and $e' = f(p')$, and there exist an operation op such that $p' \in I(op)$ and $p \in O(op)$. ω captures the set of association relations between business entities. □

Definition 9 (Business Entity Data Model). A business entity data model M is a tuple $(E, \xi^E, \varphi, \tau, \omega)$ which consists of a set business entities E and their nesting relations ξ^E, strong containment relations τ, weak containment relations φ, and association relations ω.

Given a service specification such as a WSDL file, the BE model derivation derives the BE data model for each operation provided by the service. This can

Algorithm 1. IdentifyBEandRelation

```
input: (complex) parameter $p$, parameters nesting relation $\xi^P$, business entity $e$, business entity set $E$, business entity nesting relation $\xi^E$, business entity repository $\mathcal{E}$
  /* Find a matching business entity from the repository via ontology check */
  $e' := \textsc{OntologyCheck}(\mathcal{N}(p), type(p), \mathcal{E})$
  /* Record the business entity and derive the relation with its parent entity */
  if $e' \neq \perp$ then
    for each $(p, p') \in \xi^P$ do
      $\mathcal{A}(e) := \mathcal{A}(e) \cup \{\textsc{ConvetToEntityAttr}(p')\}$
    end for
    $E := E \cup \{e'\}$
    if $e \neq \perp$ then
      $\xi^E := \xi^E \cup \{(e, e')\}$
    end if
    /* Recursively call this algorithm for each complex parameter nested in p */
    for each $(p, p') \in \xi^P \wedge \gamma(p') = complex$ do
      $\textsc{IdentifyBEandRelation}(p', \xi^P, e', E, \xi^E, \mathcal{E})$
    end for
  end if
  return $(E, \xi^E)$
```

be achieved through algorithm 1 and 2. The first algorithm generates E and ξ^E and the second refines the relation ξ^E to derive φ, τ, and ω for the operation. For each service, we iterate its operations and then loop through each complex input and output parameter of each operation to identify the BE data model. There is an overview algorithm which calls algorithm 1 to do so and the details of the algorithm can be found in our report [11].

Algorithm 1 contains three main steps. The first mainly involves the function OntologyCheck which takes name ($\mathcal{N}(p)$) and type ($type(p)$) of a complex parameter $p \in P_C$, and the business entity repository ($\mathcal{E}$) as the inputs, and returns an entity e' s.t. $e' = f(p)$. It will return nothing if there is no match found. For each complex parameter p, we check if there is a business entity that maps onto p. To do this, the service synthesis framework allows users to designate business entities for a particular context at design time. These business entities are stored in a repository $\mathcal{E}$, and p is checked against it to determine if there is a matching business entity in it. To semantically match a parameter with a business object, we assure the existence of an ontology that allows users to designate business objects for a particular context at design time. Semantic matching techniques have been well studied and this research adopts S-Match [12], a widely used semantic matching algorithm. The second step keeps the business entity and its nesting relation. If there is a mapping business entity e', parameters that are nested in p are converted and added to e''s attribute list $\mathcal{A}(e')$. The conversion involves interpreting these nested parameters as attributes of e' and skipping parameters that should not be attributes. Then e is added to *op*'s entity set E_{op}, which stores all business entities discovered in *op*. If there is

Algorithm 2. RefineBERelation

```
input: service operation set $OP$, service operation $op$ ($op \in OP$), set of business entity
  sets $\bigcup_{z \in OP} E_z$, business entity nesting relation $\xi_{op}^E$, set of mappings from parameters
  to business entities $f$
  /* Set flags for indication of whether a specific relation holds */
  $r^\tau :=\perp$ /* flag for strong containment relation */
  $r^\omega :=\perp$ /* flag for association relation */
  for each $e \in E_{op}$ do
    /* Iterate for all the business entities nested in $e$ */
    for each $(e, e') \in \xi_{op}^E$ do
      /* Iterate for all the operations manipulating $e'$ until a strong containment
      relation fails to hold */
      $X := OP$
      while $X \neq \varnothing \wedge r^\tau \neq false$ do
        Select $x \in X$ s.t. $e' \in E_x$
        $Y := E_x$
        while $Y \neq \varnothing \wedge r^\tau \neq false$ do
          Select $e'' \in Y$ s.t. $e'' \mapsto e'$
          $r^\tau := (e'' = e)$
          $Y := Y \setminus \{e''\}$
        end while
        $X := X \setminus \{x\}$
      end while
      /* Collect strong containment, weak containment, and association relations */
      if $r^\tau = true$ then
        $\tau_{op} := \tau_{op} \cup \{(e, e')\}$ /* collect strong containment relation */
      else if $e' \mapsto e$ then
        for each $z \in OP$ do
          Search for $p, p'$ s.t. $p' \in I(op) \wedge f(p') = e' \wedge p \in O(op) \wedge f(p) = e$
          if there exist such $p$ and $p'$ then
            $r^\omega := true$
            $\omega_{op} := \tau_{op} \cup \{(e, e')\}$ /* collect association relation */
          end if
        end for
        if $r^\omega := false$ then
          $\varphi_{op} := \varphi_{op} \cup \{(e, e')\}$ /* collect weak containment relation */
        end if
      end if
    end for
    /* Reset flags for relation indication */
    $r^\tau :=\perp$
    $r^\omega :=\perp$
  end for
  return $(\tau_{op}, \varphi_{op}, \omega_{op})$
```

a business entity e containing e', this relation is recorded as nesting, which is a part of *op*'s BE data model ξ_{op}^E. e is an input parameter of algorithm 1 and it refers to the parent entity of current entity e'. The final step takes every complex parameter p' that is nested in p and recursively calls the algorithm itself to process each p'. The current e' is passed on to next invocation to form the nesting relation because it is the parent of each $f(p')$. Essentially, algorithm 1 incrementally updates *op*'s BE data model until all entities and nesting relationships are processed.

Once E_{op} and ξ^{E}_{op} are identified, we can call algorithm 2 to refine the relations and derive the other three types of relations for *op*: strong containment (τ_{op}), weak containment (φ_{op}), and association (ω_{op}) to complete the BE data model for *op*. Algorithm 2 iterates each business entity e in *op*'s BE data model and then assesses the relationship between e and every business entity e' that is nested in e. This assessment is carried out according to the definitions of strong containment (Definition 6), weak containment (Definition 7), and association (Definition 8).

3.2 Service Operation Refactoring

The above algorithms show how a BE model for an operation (i.e., M_{op}) is generated. A BE data model for a service M_s is an aggregation of BE data models of all operations provided by s. Based on the service BE data model M_s, the service operation refactoring component derives the behavioural interface for s. For any business entity e derived, there are a number of operations that manipulate e and these operations can be categorised into four groups: C_e, R_e, U_e, and D_e, which denote the set of operations for creating, reading, updating, and deleting an instance of e respectively. To categorise operations provided by a service into the four modular operations of a business entity, we identified the following mapping rules.

CREATE If the invocation of an operation requires some input parameters which are actually attributes of e and returns a reference to a business entity (i.e., $key(e)$), the operation is for creating an instance of e. In other words, an operation that is designed to create a business entity e usually requires its users to pass in values for some parameters which are attributes of e. For instance, to create a shipment order, a create operation often needs to know details of shipment order such as what goods are to be shipped, where these goods are shipped from and to. As a result, the create operation should return a reference (i.e., id) of the shipment order created. There may be multiple operations designed for creating a business entity, and the set C_e is used to keep these operations.

READ If the invocation of an operation requires information of $key(e)$ and it returns the values of parameters that are attributes of the business entity e, the operation is for reading an instance of e. The set (i.e., R_e) that stores READ operations is singleton because there is usually only one operation to read an instance of e.

UPDATE If the invocation of an operation requires information of $key(e)$ and some input parameters which are actually attributes of e, the operation is for updating an instance of e.

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

We propose an algorithm that invokes each operation *op* that manipulates a business entity e and then analyses the input and output parameters according to the aforementioned mapping rules to determine the category of *op*, i.e. whether *op* is to create, read, update or delete e. As a result, the algorithm groups each *op* and adds it into one of the following sets: C_e, R_e, U_e, and D_e. The details of this algorithm can be found in our report [11]. At this stage, there could be many operations in C_e, U_e, and D_e. For example, to create a shipping order, there are a number of operations and a service consumer needs to follow certain sequence constraints, so the next step is to generate these sequences for each modular operation.

Service behavioural interfaces (i.e., protocols) depict a set of sequencing constraints and they define legal order of messages. In this paper, a behavioural interface is formalised as business entity-based behavioural model (BE model). A BE behavioural model 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.

For each $e \in E_s$, we generate BE models for its CRUD operations. For example, the model for CREATE operation defines the operations and their invocation order for creating an instance of a business entity. To derive such a model for e, we firstly retrieve entities that are strongly contained in, weakly contained in, and associated with e. When a business entity e' is weakly contained in or associated with business entity e, an instance of e' and an instance of e can be created in any order if $\lambda(e, e') = \mathit{false}$, otherwise if (i.e. $\lambda(e, e') = \mathit{true}$), an instance of e' must be created before the creation of an instance of e. If a business entity e' is strongly contained in another business entity e, an instance of e' cannot be created unless an e is instantiated. The second step is to retrieve all operations in C_e and identify their sequence through trial/error invocation. Each *op* is called and the response is analysed. If it is positive, the invocation is in sequence. Otherwise, other operations in C_e are called. This process proceeds until either all operations are in order or all operations have been checked.

Fig. 2 (b_1) and Fig. 2 (b_2) present the BE behavioural model derived based on the e_1 focused data model shown in Fig. 2 (a). Specifically, Fig. 2 (b_1) presents the model for the situation when the compulsory factor $\lambda(e, e')$ between all the entities that are related is true. As e_3 is associated with e_1, it has to be created before e_1's creation, the same can be said for e_4 and e_2. In this case, $C_{e_1} = \{op_1, op_2, op_3\}$ and we assume the sequence of creating e_1 is $op_1 - op_1 - op_2$, so the invocations of these three operations can create an instance of e_1. As e_2 is strongly contained in e_1, it can only be created after e_1's creation. Similarly, e_5 is created after e_2's creation. Fig. 2 (b_2), on the other hand, depicts the sequence derived for the situation when the compulsory factor is false. e_3 can be created concurrently with e_1's creation and e_4 can be created in parallel with

e_2's creation. The detailed algorithm for BE Behavioural model derivation can be found in our report [11].

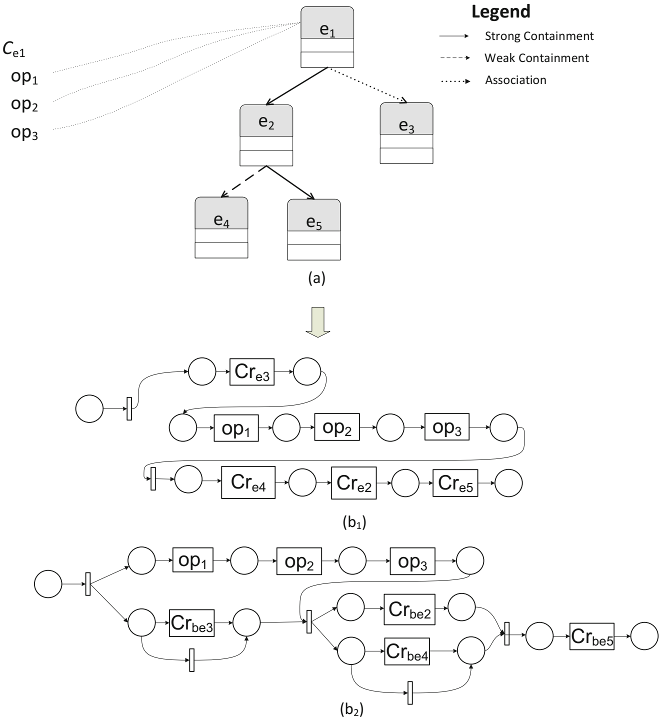

Fig. 2. An abstract demonstration for BE Behavioural model derivation

4 Implementation and Validation

To validate the service interface synthesis framework, we have developed a prototype that analyses service interfaces and generates BE data models for CRUD of a business entity. This prototype is called Service Integration Accelerator and it implements the algorithms presented in the previous section. This section presents the details of the experiments we conducted on *service interface analysis* and evaluates the framework using their results.

Hypotheses. We define three hypotheses to assess the effectiveness of the service interface synthesis framework. The first one is simplification - the service interface analysis mechanism simplifies overloaded and complicated service interfaces, so service consumers are able to utilise the structural analysis results as a guidance to facilitate their comprehension of service interfaces. As a result, this reduces the amount of time that they need in order to comprehend and then invoke services correctly. Another criteria to be examined is accuracy - the service interface analysis mechanism derives all possible business entities, however a rate of 25% of false positives is allowed as some entities may not be business entities in one specific context, but are possibly business entities in another context. Therefore, we allow service consumers to filter out those which should not be entities in a specific context. Finally, we presume that the performance fulfils our requirement - the implemented algorithms complete service interface analysis on a service within ten seconds.

Objects. Thirteen popular services (shown in Table 1) drawn from xmethods.net[3], Amazon.com, and FedEx were chosen as the experiment objects. These experiment samples were 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 services increases from IS to ES. Services in the IS category are highlighted in light grey (i.e, the first three services); Services in the SaaS category are darkgray (i.e., the four Amazon services); Services in the ES category are in dimgray (i.e., the six FedEx services).

Validation Process. We applied the Service Integration Accelerator to the interfaces of the aforementioned 13 services, and a total set of 272 operations, 12962 input parameters, and 29700 output parameters were analysed. Then, we asked the domain experts to examine the analysis results, identify false positives, and do some adjustments if necessary.

Results. Table 1 presents the detailed structural analysis results and it reports the following details: (1) the number of operations each service provides, (2) the mean number (per operation) of input parameters (IPs), output parameters (OPs), business entities (BE) derived, strong containment pairs (SCP), weak containment pairs (WCP), and association pairs (Asso Pairs), (3) The mean (per operation) of false positive rate (FPR).

According to the results, Internet services usually do not involve business entities, because they often only have a few operations with a handful of parameters. For example, the Weather Forecast service only has two operations 'GetCitiesByCountry(Country)' and 'GetForecastByCity(City, Country)'. Therefore, although the Service Integration Accelerator can pick up and present the Internet services' parameters, which provides guidance on the structural interface of these services, Internet service consumers will not benefit significantly from the analysis results because the interface is not complex.

As for services in the SaaS category, their interfaces present intermediate complexity. The number of operations provided in the four Amazon web services ranges from 9 to 157 and the average number of input parameters is between 4 and 24. There are around 3 business entities derived per operation. It may seem that service consumers can cope with this type of services as the number of input parameters for some operations is not very large, but the number of operations is quite significant and service users may find it difficult to know

---

[3] http://www.xmethods.net:5868/ve2/index.po

the sequential constraints among these operations. Having a proper structural analysis is essential to derive such constraints.

Services in the ES category are the most complex ones and they usually have a large of number of input and output parameters. Therefore, it is worthwhile to reduce complexity so that service consumers can understand the interfaces. The result shows that the Service Integration Accelerator works effectively for enterprise services. The six FedEx services in Table 1 show how these complex services are simplified. For example, the Open Shipping service has 22 operations and the average number of input parameters is 309 and the output parameter is 575. After the structural analysis, on average, we derived 11 entities per operation, which dramatically reduce the complexity as users can more easily understand the interface by looking at these business entities and their relationships. Taking the FedEx Open Shipping service as an example, the operation - 'createOpenShipment' has 1336 input parameters and 596 output parameters, by analysing these parameters, we derived 14 key business entities and their relationships as shown in Fig. 3 (b).

Regarding false positive rate, as the Service Integration Accelerator treats all complex parameters as business entities, it sometimes generates entities that should not be. For example, in the generated Amazon S3 service structural analysis result in Fig. 3 (a), 'CopyObject' and 'PutObject' should be combined as one entity, which is 'Object', and 'SetBucketLoggingStatus' should not be an entity. These false positives are 12% of total entities (32) derived. Overall, the results for majority of services experimented fulfil the hypothesis, which is 25% false positive rate, plus we allow domain experts to manually revise the business entity model and to correct these false positives. These false positives can cause the Service Integration Accelerator to derive incorrect behavioural models, but they are assessed by domain experts prior to the derivation of behavioural interfaces, so invalid ones will be prevented.

The time taken to analyse each service is not listed in Table 1 as it was fairly quick to complete the analysis, even the most complicated service - FedEx Open Shipping - took only 7 seconds, indicating the hypothesis about performance has been met.

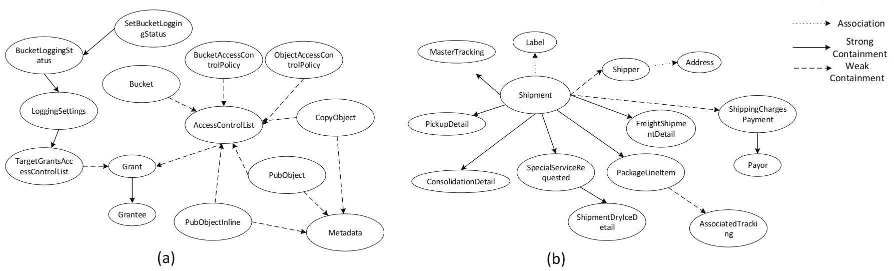

Fig. 3. Interface analysis results of Amazon S3 and Fedex createOpenshipment services

Table 1. Structural Analysis Results of the 13 services (Mean)

Services	**Operations**	**IPs**	**OPs**	**BE derived**	**SCP**	**WCP**	**Asso Pairs**	**FPR (%)**
Weather Forecast[4]	2	2	5	0	0	0	0	0
Find People[5]	3	2	1	0	0	0	0	0
MailBox Validator[6]	1	2	6	0	0	0	0	0
Amazon S3[7]	16	9	4	2	1	1	0	12
Amazon EC2[8]	157	4	8	2	1	1	1	20
Amazon Advertising[9]	9	24	243	4	3	1	0	2
Amazon Mechanical[10]	44	11	271	3	2	1	0	10
FedEx Ship[11]	5	709	239	34	40	8	1	28
FedEx Pick up[11]	3	137	41	25	23	8	1	5
FedEx Return[11]	1	20	15	3	1	0	0	0
FedEx Close[11]	6	47	18	4	3	1	1	12
Open Shipping[11]	22	309	575	11	9	3	5	24
Address Validation[11]	1	31	51	5	3	0	0	0

Open issues. Having examined the experiment results, we find that another two types of relationships between entities: inclusive and exclusive specialization. The former refers to subtypes of business entities, i.e., a set of attributes of a business entity *be* may form a new business entity be_{sub}, which is a subtype of *be*. For example, in Fig. 3 (a), 'BucketAccessControlPolicy' should be an inclusive specialization of 'AccessControlPolicy'. However, this relationship is currently considered weak containment. Inclusive specialization can also be derived using our Monte Carlo statistic based Service operation refining mechanism. That is to say, if a valid combination is a sub set of a business entity (*be*)'s attributes, this combination may become a new business entity, which is a specialization of *be*. Exclusive specialization denotes different versions of a business entity. As a business entity evolves, some parameters (i.e., features) may be added or removed, but service providers usually still support the older version of interface

---

[4] http://www.restfulwebservices.net/wcf/WeatherForecastService.svc?wsdl

[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://s3.amazonaws.com/ec2-downloads/ec2.wsdl

[9] http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

[10] http://mechanicalturk.amazonaws.com/AWSMechanicalTurk/2013-11-15/AWSMechanicalTurkRequester.wsdl

[11] http://www.fedex.com/us/web-services/

because of compatibility. Therefore, our interface analysis should be able to cope with the analysis of different version of business entities. We will examine these two types of specializations in our future work.

5 Conclusion

This paper presented a service interface synthesis framework for addressing the service interoperability challenges in the context of open and diffuse setting of global business networks. Specifically, it described a few key components of the framework, detailing service interface analysis. We also validated the framework using a variety of services. The study has demonstrated that the business entity based service interface analysis technique is an effective solution to simplify services with large, overloaded operations in interfaces. Future work will complete the framework by composing different service invocations and then validate them using a Monte Carlo statistic approach. We will also extend the prototype to support service operation refactoring to derive service behavioural interfaces and validate them.

References

1. Stollberg, M., Muth, M.: Efficient business service consumption by customization with variability modelling. Journal of Systems Integration **1**(3), 17–32 (2010)
2. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: Proceedings of the 16th International Conference on World Wide Web (WWW 2007), pp. 993–1002. ACM, New York (2007)
3. Issarny, V., Bennaceur, A., Bromberg, Y.-D.: Middleware-Layer Connector Synthesis: Beyond State of the Art in Middleware Interoperability. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 217–255. Springer, Heidelberg (2011)
4. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference, ESEC/FSE 2009, pp. 141–150. ACM, New York (2009)
5. Kumaran, S., Liu, R., Wu, F.Y.: On the Duality of Information-Centric and Activity-Centric Models of Business Processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
6. Howar, F., Jonsson, B., Merten, M., Steffen, B., Cassel, S.: On Handling Data in Automata Learning. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 221–235. Springer, Heidelberg (2010)
7. Motahari-Nezhad, H., Saint-Paul, R., Benatallah, B., Casati, F.: Protocol discovery from imperfect service interaction logs. In: IEEE 23rd International Conference on Data Engineering, pp. 1405–1409 (2007)
8. Ragab Hassen, R., Nourine, L., Toumani, F.: Protocol-Based Web Service Composition. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 38–53. Springer, Heidelberg (2008)

9. Halpin, T., Morgan, A., Morgan, T.: Information modeling and relational databases, Morgan Kaufmann series in data management systems. Elsevier/Morgan Kaufmann Publishers (2008)
10. Robert, C.P., Casella, G.: Monte Carlo Statistical Methods (Springer Texts in Statistics). Springer-Verlag New York Inc., Secaucus (2005)
11. Wei, F., Barros, A., Ouyang, C.: Deriving artefact-centric interfaces for overloaded web services (February 2015)
12. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: an Algorithm and an Implementation of Semantic Matching. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 61–75. Springer, Heidelberg (2004)