The Journal of Systems and Software 100 (2015) 149–166

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Extracting REST resource models from procedure-oriented service interfaces

Michael Athanasopoulos*, Kostas Kontogiannis

School of Electrical and Computer Engineering, NTUA, Athens, Greece

ARTICLE INFO

Article history:
Received 5 February 2014
Received in revised form 12 August 2014
Accepted 20 October 2014
Available online 29 October 2014

Keywords:
REST
Software reengineering
Service oriented architectures

ABSTRACT

During the past decade a number of procedure-oriented protocols and standards have emerged for making service-offering systems available on the Web. The WS-∗ stack of protocols is the most prevalent example. However, this procedure and message-oriented approach has not aligned with the true potential of the Web's own architectural principles, such as the uniform identification and manipulation of resources, caching, hypermedia, and layering. In this respect, Resource Oriented Architectures based on the REST architectural style, have been proposed as a possible alternative to the operation-based view of service offerings. To date, compiling a REST API for back-end procedure-oriented services is considered as a manual process that requires as input specialized models, such as, service requirements and behavioral models. In this paper, we propose a resource extraction method in which service descriptions are analyzed, using natural language processing techniques and graph transformations, in order to yield a collection of hierarchically organized elements forming REST resources that semantically correspond to the functionality offered by the service. The proposed approach has been applied as a proof of concept with positive results, for the extraction of resource models from a sizable number of procedure-oriented Web Service interfaces that have been obtained from an open service directory.

1. Introduction

Service-oriented computing has attracted significant attention as a paradigm for building interconnected software systems. In this paradigm, services are typically modeled as a set of procedural operations that encapsulate the offered functionality into distinct software units. Among all the models and frameworks that have been proposed for developing and integrating such software units, the WS-∗ stack of protocols has emerged as the de-facto standard, covering a wide spectrum of specification, deployment, and remote access concerns (W3C, 2002; OASIS, 2010). It is of no surprise that the WS-∗ stack of protocols not only received significant attention since its inception but also, served as the predominant programming paradigm for implementing, deploying and, orchestrating services in distributed networked environments. However, these protocols do not fully leverage the potential of the Web's fundamental architectural principles, as they consider the Web mostly as an infrastructure-level platform for global messaging, service access and, use (Pautasso et al., 2008; Adamczyk et al., 2011).

At the same time, the Web's global adoption and universally accepted system-wide properties and architectural traits, led a number of researchers and practitioners to look closer into the Web Architecture model in order to examine whether useful architectural abstractions could be identified, and whether these abstractions could be reused in the domain of service-oriented computing. The *Representation State Transfer* (REST) architectural style (Fielding, 2000), contains the key principles and concepts that were utilized for designing the WWW. In REST, the key information abstraction is a *resource*, which corresponds to any piece of information *that can be named* (Fielding, 2000). A resource can be accessed and manipulated through a fixed set of actions with well-defined and uniform semantics across all the resources.

As the proliferation of service computing increases to new platforms and devices, a key problem that emerges is providing different access paradigms to service capabilities. One such paradigm is resource-orientation, and it is considered important due to its architectural traits. The motivation behind the work presented in this paper, is to provide to software engineers an approach to assist them towards identifying the primary elements by which they can acquire a resource-oriented API, using as input service specifications presented in a procedure-oriented form.

* Corresponding author. Tel.: +30 2107722477.
E-mail address: athanm@softlab.ntua.gr (M. Athanasopoulos).

http://dx.doi.org/10.1016/j.jss.2014.10.038
0164-1212/

1.1. Resource-orientation and procedure-orientation

In the software engineering community there is a long standing discussion on the traits and benefits of resource-orientation and procedure-orientation as suitable architectural paradigms for service computing. Even though there is no definite answer to this discussion, and each architectural paradigm has its own merits and uses, procedure-orientation is considered as the defacto standard for service-oriented computing. However, it is recognised that there is a wealth of applications, as well as client-side components, that can benefit from a resource-oriented view of service capabilities. In this respect, a number of recent studies (Guinard et al., 2012; Pautasso et al., 2008; Pautasso and Wilde, 2009) indicate that at the architecture level, resource-orientation and REST, provide a significant capability towards addressing a number of important service computing requirements such as, ease of understanding, implementation and invocation simplicity, extensibility and, interoperability. It has also been argued that resource-oriented service design assists on addressing interface complexity issues in a more scalable way than the RPC interaction model (Vinoski, 2002; Feng et al., 2009). This is achieved by substituting the specific and separate intra-service protocols that RPC promotes, with REST's uniform interfaces. Additionally, when compared to procedure-oriented service interfaces, RESTful interfaces demonstrate a higher level of loose coupling as identified by examining various aspects of service design (Pautasso and Wilde, 2009). Finally, services exposed in a resource-oriented fashion can benefit from the properties of RESTful architectures such as, intermediate caching and the serendipitous and creative reuse of exposed resources, through the utilization of standard protocols (e.g. HTTP) and open standards (e.g. URI) (Vinoski, 2008).

Apart from architectural aspects of the procedure-orientation vs. resource-orientation discussion in service design, there are also considerations on the implementation level with regard to adopting one of the two paradigms. It has been argued (Vinoski, 2002) that the RPC interaction model imposes interface complexity and coupling, which contribute to the utilization of typically heavyweight platforms (e.g. WS-$*$ stack implementations) in order to support service realizations. However, to the fairness of RPC-based platforms, these have been developed for, and tend to address, a wealth of complex integration and life-cycle requirements (e.g. security, transaction management), for which resource-oriented frameworks currently provide limited solutions. On the other hand, given their context, RESTful HTTP services are typically based on open standards and can be implemented using standard Web technologies. In this respect, recent studies indicate that the time performance and message size of RESTful Web services are better than their SOAP and XML based counterparts (Markey and Philip, 2013; Castillo et al., 2011). Furthermore, with respect to software engineering practice, studies indicate that RESTful services are more maintainable on the server side, than the corresponding SOAP based services (de Oliveira et al., 2013), and RESTful HTTP services are easier to compose than SOAP based services (Li et al., 2012). Furthermore, new generation applications such as semantic Web applications, and applications that utilize linked data can benefit from having a RESTful API (Battle and Benson, 2008). Even though, RPC-based service orientation has its own benefits and traits for transaction-heavy systems, resource-orientation is gaining significant attention for next-generation Web applications.

Based on the above, it comes as no surprise that the number of REST-like APIs is shown to increase (Jiang et al., 2012; Leotta et al., 2012) even though these APIs are characterized by a varying conformance to REST's constraints (Renzel et al., 2012; Maleshkova et al., 2010), ranging from HTTP-tunnelled RPC to fully hypermedia-enabled RESTful interfaces. However, this increase indicates a noticeable trend towards aiming for resource-oriented exposure of service capabilities in addition to the procedure-oriented approaches which still provide a predominant style for Web APIs. In order to assess the level of maturity of an API with regard to REST, models such as the Richardson Maturity Model (Fowler, 2009) have been proposed.

1.2. Problem description and scope

In order to design a REST API, the first and fundamental step is to devise a *resource model*. A *resource model* is composed of a collection of entities, a classification of these entities into categories, such as containers, container elements, atomic elements, as well as a collection of relationships between these entities. Nevertheless, architects that opt for REST often face the challenge that the RESTful interfaces they have to design, either address functional requirements expressed in procedure-oriented terms, or have to encapsulate existing procedure - oriented functionality. In this respect, the problem of extracting RESTful resource models has been recently examined in the literature and a variety of approaches have been proposed. Most approaches proposed so far, rely on domain experts and on information models that are used by architects to devise a REST API. Such information models take the form of detailed requirements specifications, service behavioral models and data schema specifications. More specifically, in Laitkorpi et al. (2006) and Laitkorpi et al. (2009) a collection of specialized UML models that abstract structural and behavioral properties of service interfaces is used as input to compiling a resource model and consequently generating a REST API. Similarly in Liu et al. (2008) the input to compiling a resource model takes the form of E–R diagrams, class diagrams, documentation and requirements specifications. In Upadhyaya et al. (2011) language ontologies are utilized as elements to drive the compilation of a resource model while in Strauch and Schreier (2012) and Kennedy et al. (2011) wizard-like processes, driven by the user, are used to manually draft a resource model.

In this paper, we attempt to address this problem by proposing a resource model identification method that is based on the analysis of standard IDL *service description models* that are either readily available, or are easy to create. A service description model specifies the signatures of the service operations such as their names, parameters, and return types. For example, in the Web Services domain a service description is denoted by a WSDL specification file.

Here, we focus on services that follow the remote procedure call interaction model and whose interfaces are specified through WSDL documents which either explicitly use the RPC binding style, or conform to procedure-oriented designs such as the document/literal wrapped pattern (OASIS, 2010). Since WSDL descriptions can specify various aspects of a service interface, the proposed approach focuses on the *portType* element of a WSDL document, where service operations are specified, and the corresponding input and output message specifications (i.e. operation parameters). The output of the resource extraction process is a hierarchical model of resource types which captures the primary informational entities that should be present in a RESTful API for the service being analyzed. Each extracted resource type is associated with one or more CRUD operations for which there is a direct mapping to HTTP methods. However, it should be noted that the resource models extracted by the approach presented in this paper, do not address the rest of the concerns that a complete REST API specification addresses, such as resource representations, media types, hypermedia and caching. We have developed further techniques that address certain of the above concerns and can be combined with the resource extraction approach to provide enriched API specifications. However, these aspects are out of the scope of this paper.

In order for a resource extraction method to be of practical value in a production environment, we consider three design requirements namely: (a) *automation*: require as little user involvement as possible, (b) *implementation-independence*: do not require availability of source code, database schemas and generally anything more than a machine readable description of the procedural interface, and (c) *efficiency*: require that the processing time per service should be low, and thus allow for the extraction process to be incorporated in an interactive, human-driven software engineering activity.

A resource extraction process that meets the above requirements can be useful in a number of practical contexts and scenarios. Resource extraction is a primary aspect of a service interface adaptation method. More specifically, in scenarios where reimplementing and maintaining existing functionality imposes significant risks and costs, service interface adaptation is a viable solution so that, alternative paradigms such as REST can be supported. Additionally, service interface adaptation is required in contexts where service implementations are decoupled from service specification and assembling, such as in the context of Service Component Architecture (SCA). In this respect, an SCA assembler may need to expose an existing procedure-oriented service through a REST API for a component, transparently to the implementation.

Furthermore, resource extraction, along with further extraction techniques (e.g. representation extraction), can used to provide a head-start when migrating existing functionality to new implementations. Even when the final implementation includes extensions or refactorings of the exposed functionality, an automatically generated resource-oriented view of the migrant system, acquired early in migration process, can improve the overall productivity (e.g. through allowing top-down development) as well as provide insights to architects and developers as to the details of the possible resource types to be considered. Finally, since resource extraction provides a decomposition of the exposed functionality of a service primarily through CRUD operations against informational entities, it can be used in the areas of service classification, service discovery and composition where identifying the conceptual entities that underlie a service interface is a primary challenge.

The proposed approach utilizes Natural Language Processing (NLP) techniques in order to analyze service description model elements, and yield intermediate models of terms, we refer to as *Term Models*. These *Term Models* are gradually transformed using graph analysis techniques and model transformations, in order to yield a *Core Conceptual Entities Model* and ultimately a *Resource Types Model*. Consequently, the Resource Types Model can be used by a transformer to yield a WADL (Hadley, 2006) specification skeleton. This approach differs from existing methods as it requires only service signatures as input, as opposed to specialized behavioral and requirements service specification models.

The rest of the paper is organized as follows: Section 2 provides an introduction to the basic concepts of REST, outlines the proposed approach, and provides a small service that will be used as a running example to better illustrate the steps of the proposed approach. Section 3 presents the steps of the proposed resource extraction process and the corresponding techniques. Section 4 presents a case study of applying the approach for evaluation purposes to the Amazon's *Simple Storage Service* (*S3*), a widely used service the functionality of which is specified and offered both by a procedure-oriented and by a resource-oriented interface. Section 5 presents accuracy and performance evaluations through a series of experiments conducted on collections of procedure-oriented Web Services included in a public service directory on the Web, as well as, a productivity impact assessment analysis, conducted in an industrial software engineering environment. Also, Section 5 examines threats to validity and identifies limitations of the approach. Section 6 discusses prior work in the areas related to the design of REST APIs and the adaptation of procedure-oriented services to RESTful designs. Finally, Section 7 concludes the paper and provides pointers for future research.

2. Background and process outline

2.1. Resource-oriented and REST architectures

REST is an architectural style proposed by Fielding (2000). REST is defined as a co-ordinated set of the following architectural constraints: *Client-Server*, *Stateless*, *Cache*, *Uniform Interface*, *Layered System*, and the optional *Code-on-Demand*.

The first three constraints are also prevalent to the Web since its early architecture, while the next three were defined and applied as the Web's architecture evolved. The Uniform Interface constraint in particular, is regarded as a central feature in REST and it is composed of four sub-constraints, namely (a) *identification of resources*, (b) *manipulation of resources through representations*, (c) *self-descriptive messages* and (d) *hypermedia as the engine of application state* (*HATEOAS*). Through further analysis of the sub-constraints a set of twelve design criteria can be identified to facilitate the realization of RESTful designs, as discussed in Athanasopoulos et al. (2011). In this context, REST resources are defined as primary data elements representing abstractions of information that capture distinct semantics. In RESTful Web services, HTTP is typically utilized as the communication protocol, along with the URI standard that serves as a universal mechanism to express resource identifiers. More specifically, the HTTP's methods (e.g. POST, GET, PUT, DELETE) are utilized to manipulate resources (e.g. create, retrieve, update, delete) and HTTP URIs are used to identify and locate informational resources.

2.2. RESTful services at a glance

Fig. 1 provides an example of a bookstore service offered through both a procedure-oriented and a resource-oriented API. The left hand side of the picture depicts the use of a procedural service API of a bookstore service where a customer is able to search a catalog, create an order, add and remove items from the order and, submit an order. In the procedure-oriented paradigm, services are invoked by name using appropriate parameters. On the right hand side of Fig. 1, the same scenario is illustrated, but at this time is based on a resource-oriented architecture. In such a context, instead of services, there are resources such as, *bookstore*, *catalog*, *orders* collection, *order* item and *order status*. These resources are manipulated using standard HTTP operations such as, GET, PUT, POST and DELETE. For example to access a bookstore, a GET request is issued on the */bookstore* URI. This HTTP request returns, along with a 200 OK code, the representation of the *bookstore* resource which contains hypermedia elements that link to the *catalog* and *orders* resources through respective URIs (*/bookstore/catalog* and */bookstore/orders*) and *link relations*. Similarly, in order to create a new order item resource, a POST request can be issued from the client to the server pertaining to the *orders* collection resource. The result of this HTTP request is a 201 Created code along with the representation of the newly created order resource identified by the URI */bookstore/orders/1/* and containing links to the order items collection resource (*orderItems*) and the order status resource (*orderStatus*). Similarly, in order to add an item in the newly created order, a POST request is issued on the */bookstore/orders/1/items* (*orderItems* resource), and in order to update the status of the bookstore order, a PUT request can be issued to the *orderStatus* resource, referenced by the */bookstore/orders/1/status/* URI. As it can be seen from the example, the use of a resource model is a fundamental step for using REST

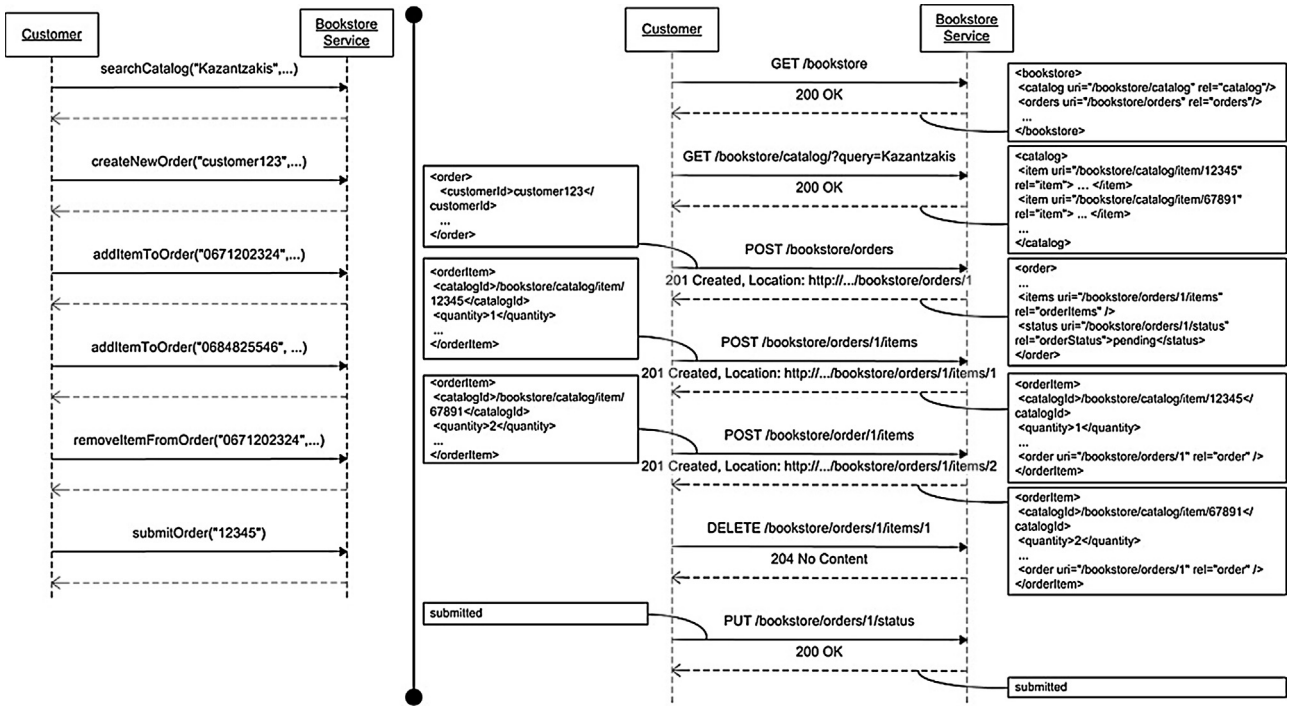


Fig. 1. Bookstore example: WS-∗ and RESTful alternatives.

and exposing a service using a REST API. Other issues include, the determination of hypermedia controls and link relations, which is outside the scope of the resource extraction phase and the focus of this paper.

2.3. Running example

In order to better illustrate the resource extraction process that we propose in this paper, we employ a running example of a simplified, yet realistic, service interface of an ordering management system (*SimpleOMS*). In a nutshell, SimpleOMS contains operations that can be used to search a catalog for items to purchase (*searchCatalog*), create and manage orders (*createOrder*, *getOrder*, *removeOrder*, *submitOrder*), add and remove items to these orders (*addOrderItem*, *removeOrderItem*), retrieve the user's submitted orders (*getSubmittedOrders*), and pay orders (*checkout*). Also, SimpleOMS supports independent shipping of order items (e.g. as soon as they become available, or sent by different manufacturers). Therefore, using the service, a user can retrieve both an item's shipping status (*getOrderItemShippingStatus*), as well as the order's shipping status as a whole (i.e. specifying whether it has been partially or fully shipped) (*getOrderShippingStatus*). All 11 SimpleOMS operations along with their signatures are listed in Table 1.

2.4. Resource extraction process outline

The resource extraction process, is a multi-step process that takes as input a set of elements included in a machine-readable interface description of the service (e.g. a WSDL document), and produces a model of potential resource types along with containment relationships between these types. The proposed approach is based on Model-Driven Engineering (MDE) principles, and includes steps that gradually transform and abstract, service interface description models fed as input, to resource models, produced as output. The resulting model denotes not only the resource types but also, a hierarchy of these types, which when combined with action semantics (e.g. HTTP methods) can serve as RESTful service interaction points. Fig. 2 depicts a high-level view of the resource extraction process that is decomposed into five basic steps, with respective models being produced. These steps are outlined below.

Step 1: Signature Models generation: *Signature Models* are introduced as means to represent procedure-oriented service interfaces in a normalized form, decoupling thus the analysis process from the specific IDL used for describing a service. This step aims to make the proposed approach applicable to other IDLs, beyond WSDL.

Step 2: Operation Terms Models and Service Terms Models generation: Once the Signature Models have been created, a collection of natural language processing techniques, and information extraction algorithms are utilized to (a) identify important service operation name terms, (b) characterize the identified terms based

Table 1
Running example: service operation signatures.

Operation	Input	Output
createOrder	auth:Auth, order:Order	result:string
getOrder	auth:Auth, orderId:string	result:Order
removeOrder	auth:Auth, orderId:string	–
submitOrder	auth:Auth, orderId:string	–
addOrderItem	auth:Auth, item:OrderItem	result:string
removeOrderItem	auth:Auth, orderId:string, itemId:string	result:string
getOrder ShippingStatus	auth:Auth, orderId:string	result: ShippingStatus
checkout	auth:Auth, orderId:string, payment:Payment	–
searchCatalog	auth:Auth, query:string	result: ArrayOfCatalogItem
getSubmittedOrders	auth:Auth	result: ArrayOfOrder
getOrderItem ShippingStatus	auth:Auth, orderId:string, itemId:string	result: ShippingStatus

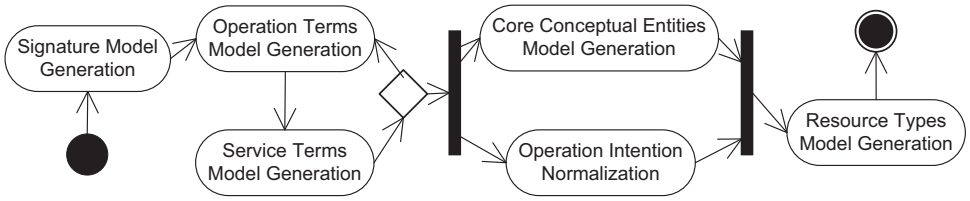


Fig. 2. High-level view of the resource extraction process.

on their role in each service operation and, (c) identify relations between the terms as a first step of creating term hierarchies.

This step aims to transform low-abstraction Signature Models that describe the interface of a service, to higher-abstraction *Operation Terms Model* (*OTM*) that contain typed terms and relationships, representing high-level views of each operation's semantics, as these can be identified by the name of the operation. Information derived from each Operation Terms Model for a given service is then combined in order to produce a single, amalgamated and more abstract service-level model, which we refer to as the *Service Terms Model* (*STM*).

Steps 3 and 4: Core Conceptual Entities Model generation and operation intention normalization: The generated Operation Terms Models and the corresponding amalgamated Service Terms Model are then utilized to (a) construct a model we refer to as *Core Conceptual Entities* (*CCEs*) and, (b) to identify dependencies between these entities. The CCEs represent the fundamental resource elements of the target REST interface. Similarly, the dependencies between Core Conceptual Entities reflect hierarchical relationships between resources elements. In this respect, the Core Conceptual Entities and their relationships provide the structure for the output resource model. In addition to CCE model generation, in Step 4, service operations are also classified into a set of normalized *intention categories* using information from the respective Signature Model and Operation Terms Model. The normalized intention categories denote whether a procedure-oriented service can be classified into one of the following categories: *Constructor*, *Destructor*, *Accessor*, *Mutator*, *Query*, *Investigator* and, *Process*. This classification is important for associating resource identifiers with HTTP verbs (e.g. POST, GET, PUT, DELETE).

Step 5: Resource Types Model Generation: The extracted CCEs and information on their associated operations is used to populate a *Resource Types Model* (*RTM*). The generated RTM is the final product of the resource extraction process.

In the following sections, we present each step of the process in detail.

3. Resource extraction

3.1. Signature Models

Signature Models are introduced as a mechanism through which service descriptions can be represented in a normalized way by abstracting any specific syntax or structure details of the original interface description language that is used as input. In this paper, we use WSDL as input to generate a corresponding normalized Signature Model. However any other IDL that exposes single services as collections of operations could be easily supported, once an appropriate Signature Model loader is provided for it. Signature Models have a simple metamodel where each service signature (s_i) has a name ($s_i.name$) which is equal to the operation name, a set of input ($s_i.input$) parameters and a set of output ($s_i.output$) parameters. A parameter (p) has a name ($p.name$), a type ($p.type$) and a multiplicity indication ($p.multiplicity$). Also, a parameter's type can be simple (i.e. a primitive type) or composite (a structure of simple or complex types). Finally, parameters may also be tagged either as *application data* or *metadata* ($p.class$), based on whether they represent information that is directly used for delivering the functionality provided by the operation (e.g. *orderId* in *getOrder* operation listed in Table 1) or, they represent information that is not directly used by the operation (e.g. authentication tokens, timestamps). This information is used in the resource extraction for the evaluation of heuristic rules. Since typical service IDLs such as WSDL do not provide information for the parameters of the service operations, a *TF-IDF* (Salton et al., 1975) categorization score is computed, in which parameters play the role of terms and signatures play the role of documents. More specifically, for each parameter p of every signature s_i, for all operation signatures S, the categorization score is computed by the C_{pf-isf} function as follows:

$$C_{pf-isf}(p, s_i, S) = pf(p, s_i) \times isf(p, s_i, S)$$

where *parameter frequency* (*pf*) is defined as:

$$pf(p, s_i) = \begin{cases} 2, & (p \in s_i.input \cup s_i.output) \wedge (substring(p.name, s_i.name)) \\ 1, & (p \in s_i.input \cup s_i.output) \wedge (\neg substring(p.name, s_i.name)) \\ 0, & otherwise. \end{cases} \quad (1)$$

and *inverse signature frequency* (*isf*) is computed by:

$$isf(p, s_i, S) = \log_2 \frac{|S|}{|\{s_i \in S : pf(p, s_i) > 0\}|}$$

Using a threshold T_{pf-isf}, a parameter p in a signature s_i can be categorized as follows:

$$p.class = \begin{cases} application\,data, & C_{pf-isf}(p, s_i, S) > T_{pf-isf} \\ metadata, & P_{pf-isf}(p, s_i, S) \leq T_{pf-isf} \end{cases}$$

A zero value for $C_{pf-isf}(p, s_i, S)$ indicates that the parameter p appears in all operation signatures (i.e. it cannot be considered specific to an operation) and therefore it can be assumed that it is a metadata parameter. However, in order to also categorize as metadata very frequent parameters that may not be present as input or output parameters in a limited subset of all operation signatures, a positive value T_{pf-isf}, close to 0, can be used as threshold. For the experiments presented in Section 5, T_{pf-isf} was set to 0.2. For instance, in SimpleOMS, the *orderId* input parameter of the *getOrder* operation (Table 1) is tagged as application data since $C_{pf-isf}(orderId, getOrder, S) = 0.65$, while the *auth* input parameter of the same operation is tagged as metadata since $C_{pf-isf}(auth, getOrder, S) = 0$.

The construction of a Signature Model for a WSDL-based interface consists of two steps: (a) loading, and (b) refinement. During the loading step, the WSDL document is processed and a first version of Signature Models is created through direct mappings

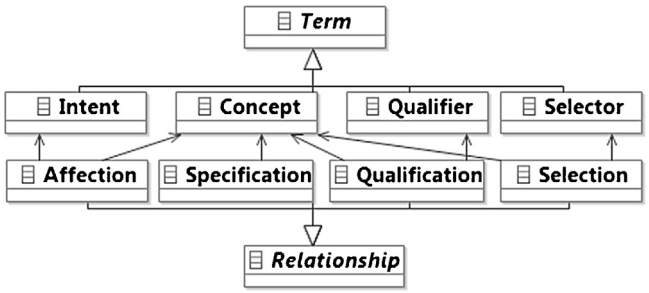


Fig. 3. Operation Terms Model metamodel elements.

between WSDL *portType* definitions denoting service operations and, Signature Model elements. During the refinement step, the Signature Models are updated based on whether the interface being adapted follows a particular style (e.g. RPC, Document), pattern (e.g. document/literal wrapped pattern), profile (e.g. WSI-BP OASIS, 2010) or, other conventions. For example, for interfaces that follow the document/literal wrapped-document pattern, "unwrapping" of request/response structures takes place in order to model the actual signature of the operation more accurately. In any case, Signature Models are merely abstractions to decouple the resource extraction process from the service description language used. In this respect, Signature Models do not affect the core steps and logic of the extraction process itself. These core steps and logic will be discussed in more detail in the following sections.

3.2. Operation Terms Models and Service Terms Models

Signature Models provide a normalized way of representing a procedure-oriented interface. However, resource extraction requires a modeling mechanism that could denote service semantics, as collections of terms, that reflect granular information as to what the meaning of an operation is. In this respect, we introduce a set of modeling elements, we refer to as *term types* and *relationships types*. Terms are substrings or concepts that can be automatically extracted by analyzing an operation name and parameters, using NLP techniques. The models capturing this information are referred to as Operation Terms Model (OTM) and Service Terms Model (STM). Operation Terms Models aim on capturing information on a single operation's functionality semantics. The proposed Operation Terms Model metamodel contains four term types and, four relationship types. Fig. 3 depicts the OTM metamodel (i.e. term types and term relationship), Tables 2 and 3 provide descriptions of the metamodel terms and relationships along with examples from SimpleOMS, while Fig. 4 depicts an instance of an OTM for the operation *getOrderItemShippingStatus*. Similarly, the Service Terms Model is defined as a labeled multigraph that amalgamates in one single model all the OTM information denoted for each operation

Table 2
OTM term types.

Term	Description	Examples
Intent	Denotes the intention of the interaction. Typically, the Intent is a verbal part of the operation name.	**add**OrderItem: Intent(**add**) **get**SubmittedOrders: Intent(**get**)
Concept	Denotes an element or an attribute with significant informational content for the logic the operation implements.	getSubmitted**Orders**: Concept(**orders**)
Qualifier	Configures or augments the semantic qualities of Concepts.	get**Submitted**Orders: Qualifier(**submitted**)
Selector	Mediates for a projection action between two Concepts (filtering, selection, etc.).	getSubmittedOrders **By**Date: Selector(**by**)

Operation Terms Model:
Operation: **getOrderItemShippingStatus**
Parsing/splitting: **get order item shipping status**
Intent: get(VB)
Concepts: order(NN), item(NN),
shipping(NN), status(NN)
Relationships:
- Affection: get AFFECTS status
- Specifications:
order SPECIFIES item
item SPECIFIES shipping
shipping SPECIFIES status

Fig. 4. Example Operation Terms Model.

of the service. The STM contains nodes that map to operation-level terms with weighted annotations representing types and weighted edges that map to relationship types. The primary role of a Service Terms Model is to serve as a means for moving from terms and relationships, to service entities and dependencies. An example Service Terms Model for the SimpleOMS service is depicted in Fig. 5.

The generation of the OTMs and STM is discussed below in more detail.

3.2.1. Operation Terms Model generation

(A) Tokenization: We focus our analysis on service operation names which are treated as identifiers with significant informational content. In a procedure-oriented service interface, operation names offer semantically rich information as to functionality provided by the operation. In other words, operation names are usually defined in a way that they can provide a short answer as to what the operation does. Typically, a tokenization task requires that one or more delimitation rules be applied to a sequence of characters (e.g. whitespace, punctuation, etc. applied in natural language texts). Such tokenization accuracy may be further improved by computational linguistics techniques that group more than one token together so that they can be treated as single tokens (for instance treating "Los" and "Angeles" as one token "Los Angeles"). These techniques are usually referred to as collocation extraction techniques (Manning and Schutze, 1999). A more general but closely related problem in software engineering is that of identifier splitting and expansion (Madani et al., 2010; Corazza et al., 2012). By reviewing related literature (Erl, 2008; Brown, 2012) and documents related to guidelines and recommended practices in service operation naming (National Cancer Institute, 2009; Rodriguez et al., 2011; Bean, 2009) and by examining a large number of service descriptions for recurring patterns, we compiled a list of frequently used operation name construction patterns. Consequently,

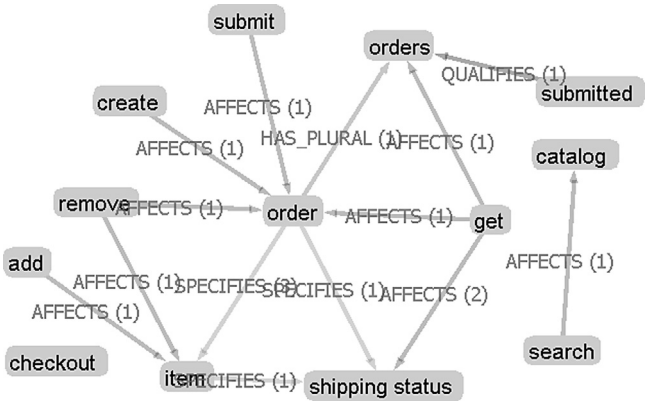


Fig. 5. Service Terms Model graph of SimpleOMS. Weights represent the total number of occurrences of a relation between two entities, from all OTMs for the service.

Table 3
OTM relationship types.

Relationship	Description	Examples
Affection ⟨Intent⟩ affects ⟨Concept⟩	Binds the Intent of an operation to one of the Concepts, indicating upon which Concept the intention of the interaction is applied.	getSubmittedOrders: get (Intent) affects orders (Concept)
Specification ⟨Concept⟩ specifies ⟨Concept⟩	It can be defined between two Concepts where the Concept specifying narrows the type spectrum of the Concept specified.	addOrderItem: order (Concept) specifies item (Concept)
Qualification ⟨Qualifier⟩ qualifies ⟨Concept⟩	Indicates a configuration or an augmentation of the semantic qualities of a Concept by a Qualifier.	getSubmittedOrders: submitted (Qualifier) qualifies orders (Concept)
Selection ⟨Concept⟩ selects ⟨Concept⟩	Indicates a projection action between two Concepts defined by a Selector.	getSubmittedOrders ByDate: date (Concept) selects orders (Concept)

we evaluated each pattern's frequency by processing a set of 867 services containing 12,918 operations, (please see Section 5.1 for more information about the dataset), in order to assess its relative significance, and our findings indicated that upper Camel Case (e.g. *GetOrders*) and lower Camel Case (e.g. *getOrders*) patterns, are the most widely adopted naming paradigms (77.7% of the cases). Mixed Camel Case with Uppercase (e.g. *getClosestATMs*) is also a frequent pattern (13.4%) even if the respective delimitation rules are more complex. Another frequent pattern is explicit delimitation (e.g. *get_submitted_orders*) (9.3%), presented mostly as underscore-based or dot-based operation name delimitation. Finally, detection of prefixes and suffixes could be also useful, to better split about 1% of operation names. Presumably, combinations of the above patterns are often used in constructing operation names.

Utilizing the acquired empirical knowledge obtained by the above analysis of operation name construction patterns, we developed a tokenization algorithm, that operates in two steps: (a) identify patterns an operation name conforms to (e.g. dot/underscore-based explicit delimitation, prefix/suffix existence, upper/lower/mixed Camel Case), and (b) perform the tokenization according to pattern-specific delimitation heuristics. The algorithm is applied in a recursive manner until no further pattern is identified, with explicit delimitation patterns being examined and tokenized first. For instance, for the operation *CreditCard_getExpirationDate* the algorithm first identifies that the operation name follows the (underscore-based) explicit delimitation pattern and splits the operation name into two parts as defined by the delimitation character. Then, pattern identification is applied again for each part. The upper Camel Case pattern is identified for the first part, and the lower Camel Case for the second part. Camel Case delimitation rules are applied to each part and tokens are acquired which do not follow any patterns and therefore the algorithm terminates.

We evaluated the accuracy of the tokenization algorithm using a simple random sampling method on the initial dataset of 12,918 operation names and for sample size $n = 388$ the tokenization accuracy was evaluated to 96.64%. The tokenization error, evaluated to 3.36%, is attributed to the following factors: (a) problematic input; for instance operation *GetUnAuthorized* is tokenized as *get*, *un*, *authorized*, (b) pattern absence; for instance, operation names *additemtolist* follows neither the explicit delimitation nor any case-related pattern, (c) pattern misidentification; for instance, operation *getATMbyLocation* is identified as mixed lower Camel Case with Uppercase and it is tokenized as *get*, *at*, *mby*, *location*.

Getting back to our running example, all operations contained SimpleOMS were tokenized through the lower Camel Case rule.

(B) Grammatical tagging: The tokens acquired through the tokenization algorithm are sequences of characters without any metadata attached for denoting the role of each token in a sequence of tokens (i.e. a service or operation name). A lightweight and effective way to achieve this is to utilize *Part-Of-Speech* (*POS*) or, grammatical tagging (Voutilainen, 2003).

POS tagging provides information on a term's grammatical usage in a sentence. For example, POS tagging indicates whether a term is used as a finite verb, a gerund, a noun, a preposition, a conjunction, and so on. In our approach, we utilize POS taggers to grammatically annotate tokens and then translate the assigned tag sequences into OTMs. Specifically, each sequence of operation name tokens is submitted to a tagger as a natural language sentence and the tagger annotates the tokens using the Penn Treebank II (PTB) tag set Bies et al., 1995. Additionally, in order to achieve a high quality POS tagging, we combined the tagging results of different POS tagging tools and namely, OpenNLP, Lingpipe and Stanford POS taggers, into one single result, through simple meta POS tagging techniques. Specifically, the meta POS tagging is performed by applying *majority voting* or the *Basic Ensemble Method (BES)* (Perrone and Cooper, 1992; Rokach, 2010) in two levels. First, on a per tagger basis, tag sequences using all available models for English – that is, two for OpenNLP, three for Lingpipe, and 4 for Stanford – are fused by majority voting. The three resulting POS tag sequences, one corresponding to each tool, are fused again by majority voting, so that a single sequence of tags is acquired. Our experimentation indicated that the above meta POS tagging technique provides higher overall accuracy when compared to any single tagger result which was evaluated to 11.46% on average.

The usage of POS tagging is an efficient alternative to incorporating machine learning techniques directly in the extraction process, which would require a domain-specific training process and respective datasets. Also, there is a wide collection of trained NLP-based models available, for a variety of languages, which makes the approach more flexible and language-agnostic.

(C) Classification of Terms and Relationships: The next step is to classify each POS tagged term into one of the four term types, and establish relations between these terms, obtained from the domain model illustrated in Fig. 3. For each operation name (i.e. tokens sequence), the sequence of grammatical tags is processed by a set of pattern-based rules to construct one term per token. The patterns are defined by the position of each tag in the sequence. Table 4 provides the list of OTM term generation patterns and corresponding examples. Then, a second set of rules is applied to generate relationships between the terms. Table 5 provides the set

Table 4
OTM terms generation examples.

Category	Description	Example
Only nouns	Assumed "get" as Intent	ItemInfo: Intent(get)
Single verb	Verb becomes Intent	getOrder: Intent(get)
Multiple verbs	Leading verb becomes Intent	runPackageBuild: Intent(run)
Noun	Noun is mapped to Concept	removeOrder: Concept(Order)
Unclassified	Unclassified becomes Concept	createDMC: Concept(DMC)
Adjective	Adjective becomes Qualifier	getTopSongs: Qualifier(Top)
Participle	Participle becomes Qualifier	getSubmittedOrders: Qualifier(Submitted)
Selector	Preposition becomes Selector	getSubmittedOrders ByDate: Selector(By)

Table 5
OTM relationships generation example.

Pattern	Relationship	Example
$[*]C_1C_2[*]$	Concept C_1 specifies Concept C_2	removeOrderItem: Order specifies Item
$[*]QC[*]$	Qualifier Q qualifies Concept C	getTopSongs: Top qualifies Songs
$I[*\setminus S]C[S*]$	Intent I affects Concept C	getOrdersByDate: get affects Orders
$[*]C_1S[*]C_2[*]$	Concept C_2 selects Concept C_1	getOrdersByLocation: Location selects Orders

of OTM relationship generation rules. For example, as illustrated in Table 5, if a *Concept* term C_1 directly precedes another *Concept* term C_2, then a *Specification* relationship is created emanating from the first term C_1 and leading to the second term C_2. Similarly, Fig. 4 depicts an OTM instance for the SimpleOMS operation name *getOrderItemShippingStatus*.

3.2.2. Service Terms Model generation

The first version of OTMs is utilized to construct a corresponding STM. This is achieved by merging the individual OTM models to an aggregate, service-level model. Specifically, STM is a multigraph that is composed of nodes that represent OTM terms, and edges that represent relationships between these terms. The STM nodes are annotated with weighted tags in the form of ⟨term type : # of term type occurrences⟩ denoting (a) the corresponding term types of the node and (b) the number of occurrences of the each type. For example, in Fig. 5 which depicts the generated STM for SimpleOMS, the term *order* is annotated (not shown) as {⟨*Concept* : 8⟩}, meaning that there are 8 occurrences of this term, tagged as *Concept*, in all OTMs for this service. Similarly, every relationship between two terms in the STM is also annotated with the number of occurrences of this relationship for the same terms, in all the OTMs for the given service. For example, in Fig. 5, the Specification relationship between the *order* and *item* terms is annotated with frequency 3, indicating that the relationship originates from three OTMs, and specifically from the OTMs generated for the *addOrderItem*, the *removeOrderItem*, and the *getOrderItemShippingStatus* operations. Using these occurrence counters, frequencies can be calculated and used by a refinement process that improves the accuracy of the created OTMs and STMs, as discussed in the next section. Finally, during STM generation an extra type of relationship named *HasPlural* is added, which connects nodes that represent the same concept in singular and plural formats.

3.2.3. Operation Terms Model refinement

The refinement step is necessary because OTMs are generated without considering the global picture of all operations containing a term, while the generated STM should provide an overall and consistent view of all terms and relationships by allowing the reassignment of the initial classification of terms, by applying a feedback loop. In particular, the refinement step analyzes the annotations in the STM, and reassigns types to the OTM model terms. For instance, assuming that the term *order* appears in 8 OTMs, and that it has been classified initially as *Concept* in the 7 out of the 8 OTMs, then the feedback loop by examining the STM could reclassify the term *order* as *Concept* in the one outlier operation, yielding thus a revised OTM for this operation where *order* is now classified only as *Concept*. This reclassification is based on a refinement algorithm that takes into account both frequency thresholds and, the feasibility of the reclassification in the examined OTM. The refinement process terminates when no further refinements are required or, the maximum number of iterations is met. More specifically, during a refinement loop, we apply the following criteria in order to select a subset of the OTMs collection as refinement candidates. More specifically an OTM is selected when: (a) STM annotations indicate that the OTM's Intent term is not classified as an Intent term in more than the 1/3 of the OTMs that it exists in, (b) STM annotations indicate that less than the 50% of an OTM's terms are classified in accordance to the majority of the classifications of the terms. For instance, supposing that the *addOrderItem* operation's terms are classified as *add: Intent*, *order: Qualifier* and *Item: Selector* and that the SimpleOMS STM annotations are *add*: ⟨*Concept* : 7⟩, *order*: ⟨*Concept* : 7⟩, ⟨*Qualifier* : 1⟩, *item*: ⟨*Concept* : 2⟩, ⟨*Selector* : 1⟩, then the OTM would be selected for refinement since 2 out of its 3 terms are not classified in accordance to the majority of the classifications as indicated by STM annotations of the respective STM nodes. Once candidate OTMs are selected, the following refinement strategy is applied. First all OTM terms are examined as potential Intent terms and based on the relative Intent frequency computed from the respective STM nodes' annotations the best one is selected as Intent. Second, once the intent is selected and possibly reclassified, all terms are aligned to the majority of the classifications for each term based on the STM annotations and their relationships are reclassified accordingly. Finally, the feasibility of the refinement is examined based on structural consistency criteria (e.g. an OTM cannot contain more than one Intent terms). Our experimentation indicates that the OTMs-STM refinement loop contributes an improvement of 7.83% to the accuracy of the generated terms models. Specifically, we evaluated the correctness of the generated individual operation models by comparing the accuracy of the generated OTMs before and after applying the STM refinement feedback loop. In this respect, in order to evaluate the refinement technique a proportional-to-size (PPS) sampling method was chosen, to account for the merging of OTMs into a service-level model. For a population of 470 services and after regarding as size the number of the operations each service contains, we followed a systematic approach in computing a sample of 370 operations (confidence level 95%, confidence interval 5%). For the examined sample, the accuracy without the feedback loop was 80.54%, while the accuracy of the produced term models when the process incorporated the feedback loop, was increased to 88.37%.

3.2.4. STM graph reduction

After the Operation Terms Model refinement loop converges and the corresponding Service Terms Model is generated, the process continues by applying a graph-processing algorithm that merges nodes together, looking for patterns in node and edge annotations, so that the intended meaning of terms as potential resource entities could be better extracted. More specifically, when node A in a STM is classified as a *Concept* and it *Specifies* a node B which is also classified as a *Concept*, and B is not *Specified* by any other node, and at the same time, node A does not have any other outgoing relationships, then A and B are merged together. For example, even if *shipping* and *status* may constitute valid terms in the context of the SimpleOMS interface, they should be considered together for better representing semantically meaningful concepts for the service's capabilities.

The new node retains all the incoming and outgoing edges to the rest of the STM's nodes. Fig. 5 depicts the STM graph for SimpleOMS after the node-merging step. The nodes *shipping* and *status* match the merging conditions mentioned above and they are collapsed into a single node (i.e. *shipping status*).

3.3. Operation intention normalization

A significant issue addressed in this step is the determination of what the general intention of a client invoking an operation is. This is done by categorizing each operation into a predefined collection of intention categories: *Constructor*, *Destructor*, *Accessor*, *Mutator*, *Query*, *Investigator* or, *Process* (i.e. not categorized in any of the

above categories). These categories are used to indicate interaction intention semantics. Thus, in this step of the resource extraction process we aim to define a mapping between an operation (e.g. *getOrderShippingStatus*) to one of these categories (e.g. *Accessor*).

The normalization technique is based on information from the operation's signature (input and output parameters) as modeled through its Signature Model, the corresponding OTM, and, pre-compiled associations between specific *Intent* terms and normalization categories. For example, an operation cannot be categorized as *Accessor* if it returns no output in the Signature Model, or *get* indicates access and *remove* indicates destruction. The pre-compiled associations were computed using the corpus of 12,918 service operations by constructing the respective OTMs and obtaining the 150 most frequent Intent terms. These 150 *Intent* terms accounted for the 90.55% of all operations and they were manually assigned to Intention categories excluding the *Process* category since an operation is assigned to the *Process* category when it cannot be categorized to any of the rest. Then, using WordNet (Miller, 1995) the initial sets of *Intent* representatives were further enriched by aggregating verb hypernym synonym sets of the *Intent* representatives. Specifically, for each *Intent* representative hypernym synonym sets whose intersection with the set of representatives was non-empty were selected and examined in order to be added to the representatives set.

The normalization technique provides increased accuracy with minimal computational overhead. We examined the results of the operation intention normalization by following a simple random sampling method on a corpus of 12,918 operations, evaluating accuracy to 88.02%. The primary contribution of the information acquired through intention normalization in the resource extraction process is in defining and evaluating resource type generation heuristics as it will be discussed in Section 3.5.

3.4. Core conceptual entities extraction

Although the terms contained in Operation Terms Models and Service Terms Models provide the required vocabulary to denote resources, they reside on a lower abstraction layer than the intended REST API resources. This distinction between terms and resources is sometimes difficult to identify; however, it is a significant aspect of the resource extraction process and leads to the introduction of an intermediate layer, that of *Conceptual Entities*. A way to illustrate the aforementioned distinction is by considering the fact that the cardinality between terms and resources can be one-to-many, since the number of entities that stem from a term depends on the context in which each term is used. For example, in SimpleOMS STM (Fig. 5) although the term *shipping status* appears only once, both *orders* and *order items* have shipping statuses, which are two semantically distinct pieces of information and should therefore relate to two distinct resource types. Additionally, since resources are regarded as interconnected elements, either structurally or by the means of hypermedia, an important aspect of the resource extraction process is to identify a hierarchy among them. More specifically, resource extraction focuses on one of the most significant and common relationships between resources, which is the existential dependency. A resource R_i is existentially dependent to a resource R_j when it cannot exist without resource R_j already existing. Such dependencies are not present in STMs; however they should exist in a resource types' model.

For the reasons discussed above, we introduce the intermediate hierarchical model of Core Conceptual Entities (CCEs) based on which, the final resource types' model will be rendered. The CCE model is defined as a labeled directed acyclic graph (DAG) each node of which has a mandatory label element that maps to a STM term. Furthermore a CCE node can be a parent of one or more CCE nodes through directed dependency edges. CCEs are identified

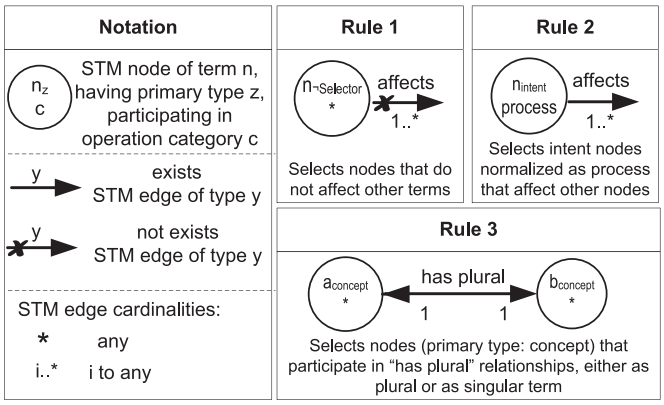


Fig. 6. CCE element selection rules. The rules are applied on the STM multigraph to select important conceptual elements.

by dependency paths from source nodes (i.e. nodes with zero in-degree) and each path defines a distinct entity. Finally, CCE nodes are characterized by a set of metadata that are computed based on the features of the underlying terms in the STM. For example, a CCE originating from a term that has an incoming *Has-Plural* relationship is tagged as *plural*. Other such metadata are *singular* and *filter*.

The extraction of CCEs is composed of two steps: *element selection* and *dependency resolution*. The element selection step specifies the subset of STM graph nodes that will be used as the corresponding CCE nodes. The dependency resolution step produces a dependency relation between the selected nodes. Element selection is performed by combining certain service-agnostic rules that operate on the Service Terms Model. Dependency resolution is performed by mapping STM relationships to CCE node dependencies and algorithmically resolving ambiguity. Both the element selection rules and dependency resolution algorithm are service and application-independent.

Fig. 6 presents the set of selection rules used. For example, Rule 1 in Fig. 6, specifies that a STM node will be selected as a CCE node, if its primary type – that is, the term type with maximum frequency for the node – is not a *Selector* and does not have an *Affects* relationship with any other STM node. Similarly Rule 2 indicates that a STM node will be selected as a CCE node, if its primary type is Intent, the corresponding operation intention is categorized a *Process* and it has an outgoing *Affection* relationship with one or more other STM nodes.

Once CCE nodes have been selected from STM nodes, a dependency resolution algorithm provides a hierarchical structure among the CCE nodes. Algorithm 3.1 outlines this process. In lines 1-8 each relationship between the selected STM elements is translated to a dependency between the corresponding CCE graph nodes. Then, the graph consisting of the selected elements and their dependencies is examined for non-trivial strongly connected components. A strongly connected component in a directed graph is a subgraph in which every node is reachable from every other node. More specifically, the *stronglyConnectedComponents* function identifies all strongly connected components of the graph defined by *N* and *D* using Tarjan's algorithm (Tarjan, 1972), and collects all non-trivial ones in the *SCC* collection. If the SCC collection is not empty ($|SCC > 0|$), the following resolution strategy is applied. First, for each identified component and for each pair of inverse dependencies it may contain, the one with lower frequency is removed (lines 10–15). The particular subgraph is checked again for strong connectivity, and provided that it remains strongly connected, an iterative process of dependency removal is applied (lines 16–21). In each iteration, the dependency that occurs most frequently in the component's cyclic paths is removed. The goal of the algorithm

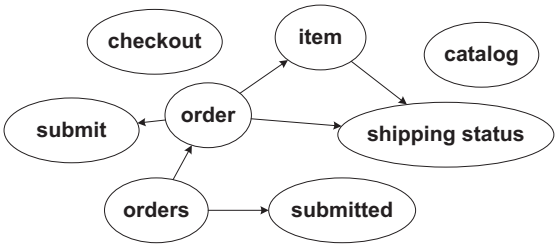


Fig. 7. Core Conceptual Entities Model for SimpleOMS.

is to remove as few dependencies as possible, respecting at the same time the following empirical prioritization of dependency significance: (a) in case of directly conflicting dependencies (i.e. dependencies between two nodes with inverse direction) the least frequent dependency will be removed as the least significant one, (b) in case of indirect dependency cycles (i.e. strongly connected components), the higher the number of cyclic paths a dependency participates in, the less significant the dependency is regarded and therefore it will be removed with higher priority. The extracted CCE model for the SimpleOMS is depicted in Fig. 7 containing 9 CCE nodes.

Algorithm 3.1 *(CCE dependency resolution algorithm).*

Require T: **STM nodes,** E: **STM edges,** N: **selected CCE nodes**
$D \leftarrow \emptyset$ # *empty dependencies set* #
for all $e \in E$ **do**
  if $type(e) \in \{Has-Plural, Selects, Affects\}$ **then**
    $D \leftarrow D \cup \{\langle stmToCCENodeMap(e.from), stmToCCENodeMap(e.to)\rangle\}$
  else
    $D \leftarrow D \cup \{\langle stmToCCENodeMap(e.to), stmToCCENodeMap(e.from)\rangle\}$
  end if
end for
while $|SCC = stronglyConnectedComponents(N, D)| > 0$ **do**
  for all $s \in SCC$ **do**
    for all $d_i, d_j \in D_s$ **do**
      if $d_i.from \equiv d_j.to$ **and** $d_i.to \equiv d_j.from$ **then**
        $D_s \leftarrow D_s \setminus \{argmin(d_i.frequency, d_j.frequency)\}$
      end if
    end for
    while s is strongly connected **do**
      $P \leftarrow s.cyclicPaths$ # *a path is a set of dependencies* #
      $I \leftarrow \bigcap D_p, \forall p \in P$ # *intersection of dependencies in paths* #
      $D_s \leftarrow D_s \setminus \{i \in I : max(p \in P : i \in p)\}$
      $D' \leftarrow D' \cup D_s$
    end while
    $D \leftarrow D'$
  end for
end while
return $DAG(N, D)$

3.5. Resource types model

3.5.1. Modeling

The final step in the resource extraction process is the generation of a Resource Types Model (RTM). The nature of a Web resource and the question of what it represents, has been the subject of discussion and debate in the Web architecture community (Berners-Lee, 2009). However, there are research efforts related to REST-based service systems that attempt to formalize several aspects of RESTful service systems using metamodeling frameworks (Selonen, 2011; Schreier, 2011). Similarly to these approaches, we regard resources as stateful informational elements that capture distinct semantics and we introduce a simple, application-neutral hierarchical metamodel for resource types. A diagrammatic view of the proposed RTM metamodel is presented in Fig. 8. Specifically, in the proposed Resource Types metamodel, an abstract *ResourceType* class may have at most one resource as its owner, and it may own a

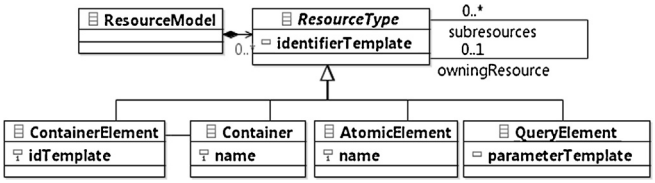


Fig. 8. Resource types' metamodel.

set of resources. Also, *ResourceType* attributes include a resource identifier template that identifies resource type classes.

The resource identifier template is composed of resource identifier fragments, which are either statically or dynamically (indicated with brackets – please see examples below) instantiated at run time. We define four concrete resource types, all of which stem from the abstract *ResourceType* class. These resource types are the following: A Container denotes a collection of resources of the same kind and may be statically named. For example, */orders* represents a collection of order resources. A ContainerElement denotes multiple resources of the same kind that belong to the same Container resource, and that they have an identification attribute that serve as a distinctive feature among each other. For example, a */orders/{order.id}* resource type represents resources that are elements of an orders' collection or, in other words, individual order resources. *AtomicElements* denote resources that represent statically-named informative entities or prefixed interaction points. For example, */orders/{order.id}/it* shipping-status represents the shipping status of an order resource which is also a resource. An *AtomicElement* may be also used to represent high-order processes attached to owner resources, or in other words manipulation actions that go beyond the fixed set of actions that the utilized communication protocol (e.g. HTTP) provides. For example, */vms/ {vm.id}/reboot* can be used to represent a resource-oriented interaction point for managing the reboot capability of a virtual machine resource. Finally, a *QueryElement* denotes resources that are selections, projections and generally, parameterized views of other resources based on combinations of parameters. For example, */orders/{ ?status, date}* is a resource type that denotes a view on the collection of orders, having a particular status and creation date.

Furthermore, there are two resource type relationships defined in the resource types metamodel, whose semantics escalate to the resource instances: (a) *is owner of* which denotes an direct existential dependency between two resources types, and (b) *is container of* which denotes a containment relationship between a Container resource type and a respective ContainerElement.

3.5.2. Comparison to existing models

The metamodel proposed in Schreier (2011) includes a broad collection of resource types. Even though there are several common concepts (e.g. *ListResourceType* as presented in Schreier (2011) and Container resource type in our RTM metamodel), several of the resource types in Schreier (2011) capture aspects that pertain to the higher-level semantics of the provided functionality, the specified parameters as well as their values (e.g. *ProjectionResourceType* or *FilterResourceType*). Such a detailed metamodel would probably require further information (e.g. runtime data) as input for the extraction process. Additionally, being an attempt to model and formalize concepts around REST, the granularity level of resource types in Schreier (2011) is probably finer than the one required for the resource extraction process. Closer to the granularity level of our metamodel is the resource model profile proposed in Selonen (2011) which was developed independently. Also, many of the resource types defined in Selonen (2011) are similar to our RTM metamodel with regard to their semantics. Nevertheless, there are

several differences that pertain to both the metamodeling definitions as well as to further constraints defined in the context of the resource model profile. For example, in Selonen (2011) *Container* resources do not have *Property* subresources. Another difference is that the QueryElement semantics are broader than the *Projection* resource type. Furthermore, in Selonen (2011) there is no support for resource types modeling high-level actions. Finally, another differentiating factor with regard to constraints is that *Projection* resources can be addressed to only by the GET method, whereas the RTM metamodel regards QueryElement resources as full life-cycle resources (e.g. they may be deleted).

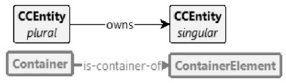
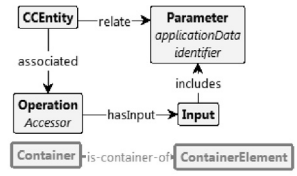
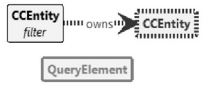
Along with WADL and WSDL 2.0 which were the first to be proposed for describing REST APIs, during the last few years, several more formats have emerged (e.g. Swagger, ioDocs, RAML, API-Blueprint), and are usually supported by tooling, as well as, are surrounded by active communities. Most of these specifications are proposed as means of providing out-of-band metadata for an API's resources, available methods and representations and therefore, they have underlying metamodels for capturing resource-oriented interfaces. It should be noted though, that dependence on out-of-band information may lead to deviation from REST's Uniform Interface constraint. Based on the fact that the approach presented in this paper is focused on the extraction of resources models, the target model of the extraction should be specified by a metamodel that focuses on types of resources and on key structural relationships between resources that can be used to organize them. In this respect, abstracting away from implementation concerns and coupling the approach to specific technologies, we chose to introduce a simple and generic resource types metamodel as a modeling mechanism that fits better to the scope of the resource extraction process. Nevertheless, the generated RTMs can be used to generate skeleton views of other formats, as it is demonstrated in Section 4 by generating WADL specifications.

3.5.3. Generation technique

The generation of the resource types model is based on the CCE models and on supplementary information from the Signature Models and term models (OTM and STM). The resource types model (RTM) generation algorithm first associates each operation to a CCE, and then iterates over the CCEs collection. For each CCE, one or more resource types are created depending on the evaluation of generation heuristics that take into account the entity's context and the associated operations. The algorithm enforces the hierarchical structure of the CCE model through resource type dependencies using the two types of relationships discussed above (is owner of, is container of). The association between CCEs and operations is done by correlating Operation Terms Models to each CCE node and its context. Then, for each CCE, the related operations are examined with regard to their Signature Model and their normalized intention category. Information about an associated operation's input and output parameters, as well as, the metadata collected for the CCE (e.g. a plural tag) are examined to determine the resource type or types of the RTM metamodel that will be populated. The list of generation rule heuristics used is provided in Table 6 and illustrated as simplified GROOVE (Rensink, 2004) rules.

Fig. 9 presents the RTM that was automatically generated for the SimpleOMS service. In the CCE model (Fig. 7), the orders CCE node is tagged as plural and the orders → order CCE is tagged as singular. This pattern leads to creating a Container resource type (/*orders*) and a ContainerElement resource type (/*orders*/{*order.id*}) associated by an is container of relationship, based on the first rule in Table 6. In the case of order items however there is no "items" (plural) CCE node, even if a similar Container-ContainerElement pair should be generated. However, the second rule of Table 6 is matched for the orders → order → item CCE, which takes into account that the CCE is associated with the operation *addOrderItem*

Table 6
Resource types model generation rules.

Rule	Description	Rule
Rule 1: C/CE pair 1	A CCE tagged as plural and its corresponding singular CCE are mapped to a Container - ContainerElement pair.	
Rule 2: C/CE pair 2	A CCE associated to an operation normalized as Constructor is expanded to a C/CE pair.	
Rule 3: C/CE pair 3	A CCE associated to an operation normalized as Accessor whose input includes an identification parameter for the CCE is expanded to a C/CE pair.	
Rule 4: Query Element	A CCE tagged as filter is mapped to a Query resource type.	
Rule 5: Atomic Element	All CCEs unmatched by the previous rules.	

which is normalized as Constructor. This fact leads to the creation of a Container-ContainerElement pair for the item CCE node (/*orders*/{*order.id*}/*item* and /*orders*/{*order.id*}/*item*/{*item.id*}). Another interesting case is the *shipping status* CCE node. There are two paths leading to it on the DAG, therefore two CCEs are identified, specifically orders → order → shipping status and orders → order → item → shipping status. The first CCE is related to *getOrderShippingStatus* operation but no generation heuristic is matched. Therefore an AtomicElement resource type is created, subordinate to /*orders*/{*order.id*} (Rule 5). The second CCE leads also to the creation of an AtomicElement. However, when placing the new resource type in the RTM hierarchy there are two resource types that it could be related to, namely /*orders*/{*order.id*}/*item* and /*orders*/{*order.id*}/*item*/{*item.id*}. In this case, the generation algorithm examines whether there are operations related to the CCE (e.g. *getOrderItemShippingStatus*) which require as input a parameter that indicates an identification element for items and relates the new resource type with the respective ContainerElement – as in our case – or, alternatively, with the respective Container.

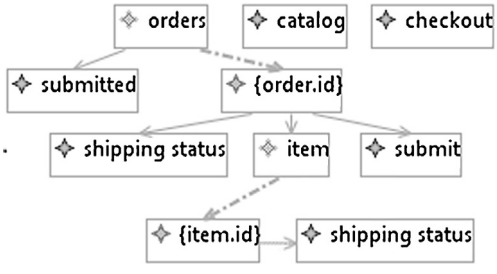
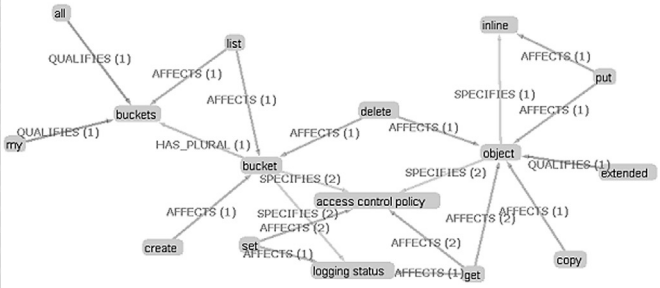


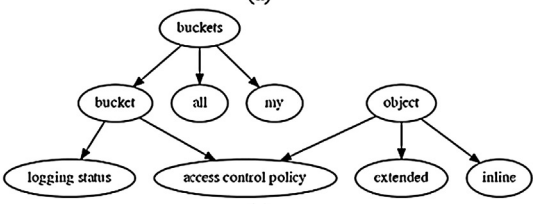
Fig. 9. Extracted resource types model for SimpleOMS service.

Operation	Tokens	POS tags	Category
CreateBucket	create bucket	VB NN	Constructor
DeleteBucket	delete bucket	VB NN	Destructor
GetObjectAccessControlPolicy	get object access control policy	VB NN NN NN NN	Accessor
GetBucketAccessControlPolicy	get bucket access control policy	VB NN NN NN NN	Accessor
SetObjectAccessControlPolicy	set object access control policy	VB NN NN NN NN	Mutator
SetBucketAccessControlPolicy	set bucket access control policy	VB NN NN NN NN	Mutator
GetObject	get object	VB NN	Accessor
GetObjectExtended	get object extended	VB NN JJ	Accessor
PutObject	put object	VBN NN	Mutator
PutObjectInline	put object inline	VB NN NN	Mutator
DeleteObject	delete object	VB NN	Destructor
ListBucket	list bucket	VB NN	Accessor
ListAllMyBuckets	list all my buckets	VB DT PRP$ NNS	Accessor
GetBucketLoggingStatus	get bucket logging status	VB NN NN NN	Accessor
SetBucketLoggingStatus	set bucket logging status	VB NN NN NN	Mutator
CopyObject	copy object	VB NN	Constructor

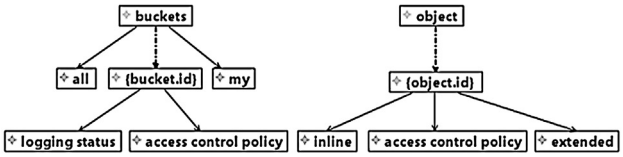
(a)



(b)



(c)



(d)

Fig. 10. Resource extraction for Amazon S3: (a) service operations with respective tokens, POS tags and intent normalization, (b) Service Terms Model, (c) CCEs model, (d) automatically generated RTM.

4. Case study: Amazon S3

As a demonstration of the proposed approach we present a case study by applying the resource extraction process to Amazon's Simple Storage Service (S3) which is a real life online cloud based data storage service. Amazon S3 offers a dual interface (i.e. a procedure-oriented and a resource-oriented interface) which allows us to better assess the extracted resource model compared to a real life one. For our analysis, we used the latest S3 WSDL document (2006-03-01), which is composed of 16 operations as depicted in Fig. 10(a).

4.1. Signature and terms models

Once the Signature Models for S3 operations are created, the tokenization algorithm is applied to each operation in order to split the operation's name into tokens. Next, the tokens get annotated with POS tags utilizing meta POS tagging by combining tag sets obtained from three POS taggers namely, OpenNLP, Lingpipe and Stanford POS. Consequently, for each operation, an OTM model is created containing terms and relationships. The tokenization and POS tagging results for the complete set of S3's operations are presented in Fig. 10(a). In parallel to the above the tool normalizes the operations' intentions yielding the results presented in Fig. 10(a) (last column). Subsequently, the 16 OTMs are merged and processed into an aggregate STM depicted in Fig. 10(b).

4.2. CCEs and resource type model

The next step is to move from terms to significant, for the service, conceptual entities. This is performed through the extraction of a Core Conceptual Entities Model. The extracted CCE model for the Amazon S3, is presented in Fig. 10(c). The model of CCEs is then used in conjunction with resource type generation heuristics, to construct the RTM for the service as depicted in Fig. 10(d). Finally, Fig. 11 presents a WADL skeleton automatically generated from the extracted RTM.

4.3. User involvement and evaluation

Presented with this output, the user may either accept it as is, or modify it. For example, in the automatically extracted Resource Types Model for Amazon S3 there is an owner relationship that can be added between the */buckets/{bucket.id}* ContainerElement and the */object* Container since, according to the service documentation, each bucket may contain one or more objects. The automatic techniques were not able to extract this relationship which can be added explicitly by the user. Also, two of the resource types that were extracted (*/buckets/all* and */buckets/my*), can be omitted because they cannot be used to map any procedure-oriented interaction, nor do they have any subordinate resource types that could be used for that purpose; therefore, they can be removed. In summary, for this service 9 out of the 11 extracted resource types were correct (0.818 precision), and all 9 expected resource types were identified (1.0 recall). Finally, RTM_{sim}, an ontology-based similarity measure which will be presented in the next section, was evaluated to 0.636. Amazon S3 provides a well-defined, content-oriented interface which is a good candidate for illustration purposes. However, a generalized assessment of the proposed approach, for a wide collection of services, is discussed in the following section.

5. Evaluation

In this section, we present results from evaluation experiments we conducted in order to assess the accuracy of the obtained results, and the performance of the extraction process using services from the ProgrammableWeb open service directory. Our evaluation experiments focus in four areas. The corresponding results are discussed in the following sub sections. More specifically, the first area of evaluation deals with the accuracy of the intermediate steps of the resource extraction process as presented in the previous sections. The results are depicted in Table 7. The second area deals with the quality of the extracted resource models, as these were compared with models created by experts. For this purpose, we have obtained a sample of 70 services that has

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:Application xmlns="http://wadl.dev.java.net/2009/02" xmlns:ns2="http://wadl.dev.java.net/2009/02/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <grammars>
        <include href="RTMM.xsd"/>
    </grammars>
    <resources>
        <resource type="Container" path="/buckets/">
            <method name="POST">
                <request>
                    <doc title="Normalization for &quot;CreateBucket&quot; back-end operation"/>
                </request>
            </method>
            <method name="GET">...</method>
        </resource>
        <resource type="ContainerElement" path="/buckets/{bucket.id}/">
            <param name="bucket.id" style="template" type="xs:string"/>
            <method name="DELETE">
                <request>
                    <doc title="Normalization for &quot;DeleteBucket&quot; back-end operation"/>
                </request>
            </method>
            ...
        </resource>
        <resource type="AtomicElement" path="/buckets/{bucket.id}/logging-status/">...</resource>
        <resource type="AtomicElement" path="/buckets/{bucket.id}/access-control-policy/">...</resource>
        <resource type="Container" path="/buckets/{bucket.id}/object/">...</resource>
        <resource type="ContainerElement" path="/buckets/{bucket.id}/object/{object.id}/">...</resource>
        ...
    </resources>
</ns2:Application>
```

Fig. 11. WADL skeleton generated from the extracted Amazon S3 RTM.

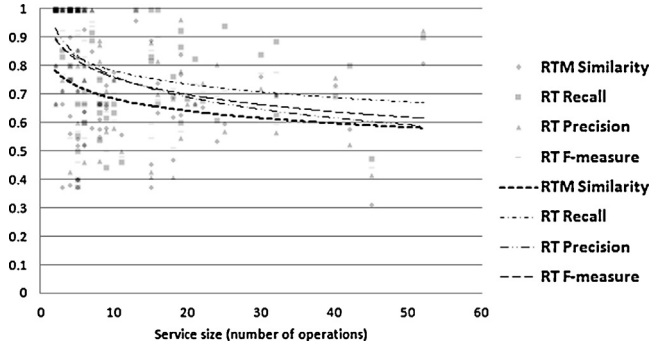


Fig. 12. Evaluation of accuracy vs. service size.

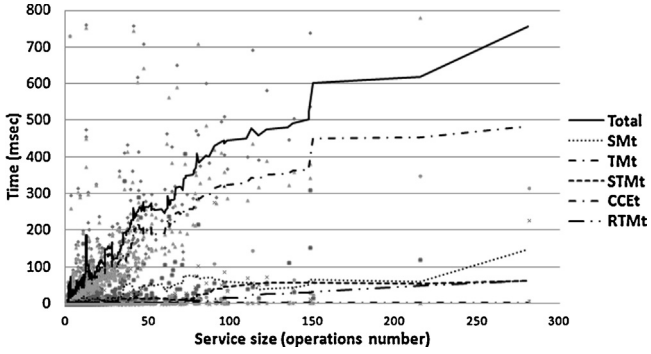


Fig. 13. Performance evaluation: resource extraction time.

served as input for the compilation of 70 Resource Types Models by experts, which models then were considered as a golden standard for analysis purposes. The evaluation treated the process as an information retrieval problem and measured *recall*, *precision*, *F-measure*, and ontological similarity between the resource model obtained by the process and the model drafted by the experts. The results are depicted in Table 8 and Fig. 12. The third area deals with the productivity enhancement by using the extraction process versus manually drafting a comparable resource model. The results are depicted in Table 9. The fourth area deals with the time performance of the different steps of the process as a function of service size. The results are depicted in Fig. 13. The following sections discuss these results in more detail.

Table 8
Experiments results: accuracy evaluation.

70 services sample	Avg	Min	Max	SD
RT_{recall}	0.796	0.375	1	0.186
$RT_{precision}$	0.785	0.409	1	0.191
$RT_{F-measure}$	0.777	0.442	1	0.167
RTM_{sim}	0.697	0.313	1	0.187

Table 9
Evaluation of productivity impact.

Service	R_e	R_t (s)	C_e	C_t (s)	I_e	I_t
#1	9	142.1	36	568.5	75.0%	75.0%
#2	13	340.5	30	736.3	56.7%	53.8%
#3	2	8.5	12	50.8	83.3%	83.3%
#4	6	56.5	15	98.2	60.0%	42.4%
#5	12	114.8	20	191.0	40.0%	39.9%
Avg					63.0%	58.9%

Table 7
Evaluation of intermediate extraction steps.

Step	Dataset	Sampling method	Sample size	Accuracy
Tokenization	12918 op.	Simple random	388	96.6%
Operation Terms Models generation	12918 op. from 867 serv.	PPS & systematic	370	80.54%, 88.37% (feedback)
Intention normalization	12918 op.	Simple random	388	88.02%

5.1. Experiments design and data set

Most of the approaches discussed in the related work Section 6, require either information beyond a machine readable definition of an interface, or require significant user involvement in the resource discovery process. Furthermore, to our knowledge there are no standard datasets available for resource extraction to be used for a comparative assessment of the proposed approach. In this respect, we have opted for a dataset acquired through the ProgrammableWeb service directory by selecting SOAP-based services with valid procedure-oriented WSDL interface descriptions. Data collection was performed using a tool that was built for this purpose. The tool used the directory's API to collect entries for SOAP-based Web Services and retained the ones that included WSDL URIs in their descriptions. Then, the collected URIs were used to retrieve WSDL documents, each of which was validated before being added to the dataset. In this way, 867 valid WSDL documents containing 12,918 operations were collected and they were used for conducting evaluation experiments. The list of the 867 WSDL URIs as well as further datasets used in the evaluation phases are available online.[1]

5.2. Accuracy of intermediate extraction steps

The first set of evaluation studies relates to the accuracy of the individual steps of the process. The results for this evaluation are depicted in Table 7. More specifically, by using applicable sampling methods for each step (Simple Random, PPS, Systematic), groups of operations were selected from a total of 12,918 operations. The results of each step of the automated process were evaluated by experts, as to how close they were on artifacts that could manually craft themselves. The results indicate that the intermediate models are of high accuracy, with values ranging from 88% to 96.6%.

5.3. Accuracy of resource extraction process

The evaluation of the accuracy of the obtained resource model was performed using a sample of 70 randomly selected WSDL documents out of the initial set of collected WSDLs, excluding single-operation services. The average service size for the set of selected services was 11.73 operations per service (min: 2, max: 52, SD: 11.2), and it included services from several domains such as e-commerce, cloud hosting, image processing, telecommunications, investments to name a few. Every selected service was examined by two software engineers working independently, who built respective Resource Types Models. The two manually built RTMs for each service, were comparatively examined and merged into a final model. The experts that participated in creating the dataset had significant experience with the REST architectural style and RESTful HTTP services. Due to limited expert person power in our disposal, we limited the size of services interfaces to manually analyze and process to a total of 70 service interfaces. Nevertheless, a larger dataset could further improve the statistical significance of the obtained results.

Since the proposed approach is an extraction process, we have evaluated measures related to precision, recall and *F*-measure for the extracted resource types, reporting average, minimum, maximum and standard deviation values. Additionally, viewing a Resource Types Model as an ontology of resources, the measure of minimum weight maximum graph matching distance (MWMGM) (David and Euzenat, 2008) defined for ontology distances, was utilized for measuring extraction correctness when computed between the extracted models and the manually built ones. The MWMGM distance is defined as a distance measure based on a dissimilarity function δ between ontological entities, which in RTMs' case are the resource types. Specifically, we define RTM similarity $RTM_{sim} : E \times S \rightarrow [0, 1]$ as an evaluation measure for resource extraction techniques, where E is the set of extracted RTMs and S the set of manually created RTMs, computed as follows:

$$RTM_{sim} = 1 - \sum_{i}^{n} \frac{\Delta_{mwmgm}(e_i, s_i)}{n}$$

where n is the sample size, $e_i \in E$, $s_i \in S$ and Δ_{mwmgm} is the MWMGM distance of a pair of RTMs. The dissimilarity function $\delta : e_i \times s_i \rightarrow [0, 1]$ used for RTM_{sim} is based on the Levenshtein distance between resource type identifier templates, taking into account identifier fragments. Also, the minimum weight maximum matching M can be computed utilizing the Hungarian algorithm (Kuhn, 1955), extended in order to deal with $n \times m$ cost matrices. RTM_{sim} not only evaluates the level of matching between individual resource types but also takes into account the organization of RTMs, providing thus a reliable accuracy assessment.

5.3.1. Recall, precision, F-measure, similarity

The section discusses results related to the accuracy of the produced Resource Types Model. More specifically, we evaluated the resource type precision ($RT_{precision}$), resource type recall (RT_{recall}), resource type *F*-measure ($RT_{F-measure}$), and Resource Types Model similarity (RTM_{sim}) between the automatically extracted models, and the manually crafted ones for a sample of 70 services and by computing average, min, max and standard deviation values. Table 8 depicts the results obtained from experiments by comparing the outcome obtained by the automated process and the manually crafted models by experts for the sample services. In summary, the average RT_{recall} was evaluated to 0.796, the average $RT_{precision}$ was 0.785, the average $RT_{F-measure}$ was evaluated to 0.777 and the average RTM_{sim} was 0.697. Given the diversity of the randomly selected service definitions and the complexity of the task, the accuracy of the proposed approach is considered very positive. Another observation of our analysis is that, as service size increases the values of all accuracy metrics decrease (Fig. 12). However, RT_{recall} decrease with a lower rate than the rest measures. This is related to the fact that since as more operations in a service are analyzed, more terms and relationships are added to the STM graph. On the one hand, RT_{recall} is primarily dependent to the CCE selection step whose performance is rather stable. On the other hand, as more relationships are added to the STM, more dependencies are created between CCEs, which may lead to the creation of imprecise resource type hierarchies. These redundant hierarchies however are usually easy to identify and the user can trim the RTM without significant effort. Finally, it is noted that RTM_{sim}, which takes into account the hierarchical aspect of the Resource Types Models has consistently significant values and it can be used to demonstrate how the level of accuracy for the resource extraction process varies over service size.

5.3.2. Computational performance and scalability

Computational performance evaluation was based on applying the resource extraction approach to the full set of the 867 services (12,918 operations). Computational performance and scalability was examined through measuring the resource extraction time (REX_t) and its decomposition into Signature Model construction time (SM_t), term models generation time (TM_t), STM refactoring time (STM_t), CCE extraction time (CCE_t), and RTM generation time (RTM_t) are evaluated with regard to service size. The prototype run on a 2-core CPU 2.8 GHz, 4 GB RAM workstation. Fig. 13 illustrates the moving average of resource extraction time required,

[1] http://www.softlab.ntua.gr/~athanm/resourceExtraction

along with its decomposition into SM_t, TM_t, STM_t, CCE_t, and RTM_t as described above. The process, without any particular optimization, runs in less than a second for each service, for the majority of services, scaling almost linearly compared to service size. Also, most time is spent in the generation of OTMs and STM (Term Models) that includes the time spent for getting the POS tag sequences from the tagging tools. Therefore, it can be said that the process is lightweight enough to be easily integrated in a real-time, interactive process.

5.4. Refinement vs. construction: impact on productivity

As discussed above, once the RTM is generated the user can review and refine the model in order to make sure that it truly reflects the examined service interface. In order to assess the usefulness of the approach in the context of an enterprise production environment, we invited a software architect and his team of software engineers to apply the proposed resource extraction approach to a set of procedure-oriented Web Services. The resource extraction prototype was used to analyze five services and apply the resource extraction process. Once each resource types model was extracted, the engineer driving the process examined and revised it, so that it reflected his mental model of the service. During this refinement process the RTM editor captured and logged the creations, deletions and modifications of model elements, along with a timestamp associated to each action. Using the RTM editor logs and the user-refined RTM models we performed an evaluation analysis to compare the effort and time required to refine each model (R_e and R_t, respectively) against an estimation of the effort and the time that would be required to build the revised model from scratch (C_e and C_t). R_e is equal to the number of creation, deletion and modification actions during the user's refinement session. R_t is equal to the total time required for the refinement session in seconds. For computing C_e we used the strictest possible estimation by considering the minimum number of actions required to build the refined model from scratch which is equal to number of creation actions for all nodes and relationships contained in the refined model. C_t was computed in a similar way through multiplying the average time required for creation and modification actions during the refinement with C_e. It should be noted that during the creation of a resource model from scratch, it is typical for a user to add model elements that he/she then removes or modifies, before reaching the final desired model. Nevertheless, C_e and C_t measures reflect the best possible scenario for the manual RTM creation case, assuming that no such corrections are required and minimizing thus the respective values.

Using R_e, R_t, C_e and C_t we can now compute effort impact I_e and time impact I_t as: $I_e = (1-(R_e/C_e)) \times 100\%$ and $I_t = (1-(R_t/C_t)) \times 100\%$. Impact values reflect the percentage of effort and time that is avoided (positive values denote benefit) or added (negative values denote overhead) by utilizing the proposed extraction technique and then refining the extracted models instead of going through manual construction from scratch. Table 9 presents the results of the experiments for assessing productivity impact. In a nutshell, using the prototype the practitioners' team was able to acquire resource models for their procedure-oriented services with 63% less actions on average and in 58.9% less time than what would be required through a manual process.

5.5. Threats to validity and approach limitations

Focusing on the accuracy evaluation study we identify the following threats to validity.

Internal validity: As described in Section 5, due to the lack of available datasets that could be used as golden standards in order to evaluate the accuracy of the approach, we resorted to inviting expert engineers to manually create RTM models for the 70 randomly selected procedural service interfaces. Nevertheless, the expert engineers who created the RTM models were not the original designers of the service interfaces, and most of the times, they had no prior knowledge of the interfaces they analyzed. Furthermore, only a subset of the examined interfaces provided documentation and for those services that documentation was available, it varied in size and quality. In this respect, a threat is the extent to which the engineers who created the standard models managed to comprehend the initial service interfaces and translate them to correct resource-oriented representations.

External validity: Web services published by different service providers are diverse with regard to naming conventions. This is because no central authority governs and enforces horizontally such design decisions; at the same time, interoperability specifications do not generally regard such concerns inside their scope. Additionally, it is a common practice to have service interfaces descriptions be generated in a bottom-up fashion. In these cases, tools assist developers in exposing service implementations by automatically generating artifacts such as service interface descriptions. Therefore, the naming and structural patterns followed in service interfaces generated in bottom-up fashion are affected by the implementation of the service, as well as by the tools that are used to generate the interface specifications. The proposed approach attempts to address the diversity problem by separating different concerns of the resource extraction challenge into distinct, well-defined steps. In each step, certain techniques and rules are proposed which, as examined in Section 5 provide positive results with regard to accuracy as well as performance. In this respect, the proposed techniques are designed against diverse contexts and attempt to render as accurate models as possible based on the primary assumptions of consistent, human readable and understandable naming conventions across the service interface specification. Nevertheless, a diverse environment is not always the case. For instance, there may be cases of highly customized or poorly designed service interfaces which will have to rely on external artifacts, or even on human explanation to convey the semantics of the interface elements. In this respect, in contexts like the ones described above the accuracy of the approach is expected to be lower, unless at least tokenization and term model generation steps are adapted to the specific environment. Whether such adaptation effort is justifiable depends on the extend of the number and the volume of services to be analyzed, which can range from a single service to whole repositories of hundred of services. For larger repositories an automated extraction approach such as the one presented in this paper may be preferable.

Additionally, being a multistep process there are possible failure scenarios for each step. We present certain such indicative scenarios below along with applicable remediation practices that can be followed:

- Wrong tokenization has direct impacts to OTM and STM generation. Wrong tokens are typically unknown words for the POS taggers and therefore, the accuracy of the POS tag sequence may be affected. Typical cases of wrong tokenization are discussed in Section 3.2. Nevertheless, based on the experiments conducted, the tokenization error is low and even when wrong tokenization occurs, it affects only a subset of the terms, leading to a reduced accuracy but not to complete failure of the overall process.
- The limited grammatical and terminological scope of operation names as well as the *morpho-syntactic ambiguity* that is known to inherently affect POS tagging, may lead to imperfect OTM models, even if the tokenization step was accurate. The problem is partially addressed by utilizing the meta POS tagging technique

as described in Section 3.2, where different taggers are utilized to combing POS tagging results through majority voting data fusion. Furthermore, the OTMs-STM refinement loop also discussed in Section 3.2 ensures increased accuracy of OTM generation and consequently of STM generation. Experiments indicated an accuracy improvement of 7.83% attributed to the refinement loop.

- RTM generation errors can be attributed primarily to operation intention normalization error and parameter tagging error. Such errors can result in an imperfect RTM model and more specifically, in RTM models that have inaccurate hierarchies or missing resources. However, being the last step of the process, the user can examine the result and directly remediate the error of this step.

Finally, the approach is extensible so that new heuristics can be incrementally added if needed to any step in order to improve accuracy.

6. Related work

The challenges and issues related to the extraction of resource models and the compilation of REST APIs from procedure-oriented services have recently attracted the attention of both academics and practitioners (Athanasopoulos and Kontogiannis, 2010; Kennedy et al., 2011; Laitkorpi et al., 2006, 2009; Liu et al., 2008; Upadhyaya et al., 2011; Strauch and Schreier, 2012). More specifically, in Laitkorpi et al. (2006) a UML-based approach to abstract legacy APIs into a canonical interface model that can be used to expose REST-like resources is introduced. The method proposed in Laitkorpi et al. (2006) requires a set of UML models that describe in detail the structural as well as, behavioral aspects of the interface. The UML models required, go beyond typical IDL-based interface descriptions, they are not usually available in practice, and they would require considerable effort to build. A subsequent work presented in Laitkorpi et al. (2009), describes a model-driven process for gradually transforming procedure-oriented specification models (e.g. a Sequence Diagram of top-level components) to resource-oriented interfaces. However, significant user involvement is assumed since the user has to manually translate the procedure-oriented specifications to an information model that is subsequently used to generate the resource model. Our approach attempts to automate the information extraction challenge involved in resource extraction, limiting significantly user involvement. Another approach is presented in Liu et al. (2008), where the authors propose a process for reengineering legacy systems to REST. The approach begins by analyzing the source code of the system. Also, it requires analysis of other artifacts such as ER models, class diagrams as well as, requirements and documentation models. Our approach, on the contrary, is implementation-agnostic and does not assume availability or access to the implementation of the service. Migrating SOAP-based services to REST-based services is discussed in Upadhyaya et al. (2011). Focusing on the resource extraction process, the approach in Upadhyaya et al. (2011) grounds its analysis on segmenting the operations signatures set into clusters. Consequently, words included in the operations of each cluster are processed utilizing WordNet's hyperonym-hyponym relationships and heuristics to form resource identifiers. Our approach does not require the use of language ontologies such as WordNet to correlate terms and furthermore, relationships between terms stemming from service descriptions can be arbitrary and do not require a hypernym-hyponym structure. Also, in Upadhyaya et al. (2011) the clusters formation is dependent to the existence of multiple operations related to a term, implying thus a CRUD-oriented interface design. Such an assumption may not hold in many service descriptions in practical scenarios. Finally, several other approaches that engage the user in a wizard-like extraction process have also been proposed Kennedy et al. (2011), Voutilainen (2003). The major difference of Kennedy et al. (2011), Voutilainen (2003) from our approach is that they are driven by, and require user input at each step.

In our previous work Athanasopoulos and Kontogiannis (2010), we presented a technique for identifying resources from legacy service descriptions, through structurally analyzing operation signatures. More specifically, in Athanasopoulos and Kontogiannis (2010) input and output message structures were analyzed in order to extract template resource identifiers. This analysis required the use of metadata that were assumed available by the user, pertaining to the semantic classification of WSDL elements (e.g. a parameter classified as a Container). The approach presented here differs from Athanasopoulos and Kontogiannis (2010) as (a) it does not assume such metadata information be available, (b) it relies solely to WSDL specifications to extract models of resource types and finally, (c) it examines services operation signatures for identifying resources beyond their structural level.

Finally, in Kopecky et al. (2008) and Sheth et al. (2007) the challenge of bridging the semantic distance between RESTful services and procedure-oriented designs is examined from a different angle. Specifically, hRESTS proposed in Kopecky et al. (2008) is a microformat specification that can be used to identify REST interaction points as operations with input/output parameters, while the specification can take the form of annotations embedded to interface descriptions. SA-REST (Sheth et al., 2007) extends hRESTS through further annotations which allow for the specification of service facets (e.g. supported data formats, language bindings, etc.). Even though theses approaches may seem to take the opposite direction, that is identifying operations through resource-oriented functionality decompositions, if desired they can be used complementary to a resource extraction approach. More specifically, hRESTS and SA-REST can be used to annotate the results of a resource extraction process with information that traces back to the procedure-oriented interface so that associations between the extracted information and the input information are explicitly provided.

7. Conclusion and future directions

The extraction of a resource model and the compilation of REST APIs from procedure-oriented services has been considered as a task that involves significant manual effort and specialized input artifacts. In this paper, we proposed an approach that utilizes information extraction techniques and model transformations in order to automate to a large extent the resource extraction process. Due to the heuristic nature of the transformations used, we attempted to objectively evaluate the extraction results using both typical Information Retrieval metrics as well, as a formal similarity measure between automatically and manually obtained results based on the minimum weighted maximum graph matching ontology distance.

The proposed approach is an improvement compared to existing approaches as it does not require initial models to be crafted by the users, and provides a tractable process for extracting hierarchical models of resource types that can be used for generating a REST API. We have applied this approach to a large collection of real-life services obtained from the ProgrammableWeb open repository with positive results. Nevertheless, this work opens new possibilities for future research directions. More specifically, future research can proceed in three main directions. The first direction deals with enhancing the performance and accuracy of

the resource extraction process. In this respect, machine learning techniques, rather than POS tagging translation rules can be investigated in order to extract domain-specific operation models and assess whether these machine learning techniques perform better than the current approach. Another possibility is to investigate the analysis of informal information such as external or internal documentation of the service IDL. The second direction pertains to extensions of the overall framework by investigating techniques that allow for the attachment of hypermedia relations to the extracted resource types models. These hypermedia relations can then be used from a runtime to drive the interaction between a client and a server utilizing the extracted REST resources. Finally, the third direction is to investigate techniques by which such an IDL analysis can be leveraged in other contexts beyond resource extraction. A possible approach is to analyze IDL service specifications in order to extract collections of meaningful terms and concepts that can be further used to group or cluster collections of semantically similar services in an organization that has a large and diverse portfolio of services.

Acknowledgements

This work is supported by IBM Canada CAS Research under the Research Fellowship Project no. 754.

References

Adamczyk, P., Smith, P.H., Johnson, R.E., Hafiz, M., 2011. Rest and web services: in theory and in practice. In: REST: From Research to Practice. Springer, New York, pp. 35–57.

Athanasopoulos, M., Kontogiannis, K., 2010. Identification of rest-like resources from legacy service descriptions. In: 2010 17th Working Conference on Reverse Engineering (WCRE), IEEE, pp. 215–219.

Athanasopoulos, M., Kontogiannis, K., Brealey, C., 2011. Towards an interpretation framework for assessing interface uniformity in rest. In: Proceedings of the 2nd Intl Workshop on RESTful Design, ACM, pp. 47–50.

Battle, R., Benson, E., 2008. Bridging the semantic web and web 2.0 with representational state transfer (rest). Web Semant.: Sci. Serv. Agents World Wide Web 6 (1), 61–69.

Bean, J., 2009. SOA and Web Services Interface Design: Principles, Techniques, and Standards. Morgan Kaufmann, Burlington.

Berners-Lee, T., August, 2009. A Short History of 'Resource' in Web Architecture. http://www.w3.org/DesignIssues/TermResource.html

Bies, A., Ferguson, M., Katz, K., MacIntyre, R., Tredinnick, V., Kim, G., Marcinkiewicz, M.A., Schasberger, B., 1995. Bracketing guidelines for treebank II style penn treebank project. University of Pennsylvania.

Brown, P.C., 2012. Architecting Composite Applications and Services with TIBCO. Addison-Wesley Professional, New Jersey.

Castillo, P.A., Bernier, J.L., Arenas, M.G., Guerdon, J.J.M., Garcia-Sanchez, P., 2011. Soap vs rest: comparing a master-slave GA implementation, CoRR abs/1105.4978.

Corazza, A., Di Martino, S., Maggio, V., 2012. Linsen: an efficient approach to split identifiers and expand abbreviations. In: 2012 28th IEEE Intl. Conference on Software Maintenance (ICSM), IEEE, pp. 233–242.

David, J., Euzenat, J., 2008. Comparison between ontology distances (preliminary results). In: The Semantic Web-ISWC 2008, Springer, pp. 245–260.

de Oliveira, R.R., Sanchez, V., Vinicius, R., Estrella, J.C., Pontin de Mattos Fortes, R., Brusamolin, V., 2013. Comparative evaluation of the maintainability of restful and soap-wsdl web services. In: 2013 IEEE 7th International Symposium on the Maintenance and Evolution of Service-oriented and Cloud-based Systems (MESOCA), IEEE, pp. 40–49.

Erl, T., 2008. SOA Design Patterns. Pearson Education, Boston.

Feng, X., Shen, J., Fan, Y., 2009. Rest: an alternative to RPC for web services architecture. In: First International Conference on Future Information Networks. ICFIN, 2009, IEEE, pp. 7–10.

Fielding, R.T., 2000. Architectural Styles and the Design of Network-based Software Architectures (Ph.D. thesis). University of California.

Fowler, M., 2009. Richardson Maturity Model: Steps Toward the Glory of Rest. http://martinfowler.com/articles/richardsonMaturityModel.html

Guinard, D., Ion, I., Mayer, S., 2012. In search of an internet of things service architecture: REST or WS-*? a developers perspective. In: Mobile and Ubiquitous Systems: Computing, Networking, and Services. Springer, pp. 326–337.

Hadley, M.J., 2006. Web Application Description Language (WADL), Technical Report.

Jiang, W., Lee, D., Hu, S., 2012. Large-scale longitudinal analysis of soap-based and restful web services. In: 2012 IEEE 19th International Conference on Web Services (ICWS), IEEE, pp. 218–225.

Kennedy, S., Stewart, R., Jacob, P., Molloy, O., 2011. Storhm: a protocol adapter for mapping soap based web services to restful http format. Electron. Commer. Res. 11 (3), 245–269.

Kopecky, J., Gomadam, K., Vitvar, T., 2008. hrests: an html microformat for describing restful web services. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. WI-IAT'08, Vol. 1, IEEE, pp. 619–625.

Kuhn, H.W., 1955. The Hungarian method for the assignment problem. Naval Res. Logist. Q. 2 (1-2), 83–97.

Laitkorpi, M., Koskinen, J., Systa, T., 2006. A UML-based approach for abstracting application interfaces to rest-like services. In: 13th Working Conference on Reverse Engineering. WCRE'06, IEEE, pp. 134–146.

Laitkorpi, M., Selonen, P., Systa, T., 2009. Towards a model-driven process for designing restful web services. In: IEEE International Conference on Web Services. ICWS 2009, IEEE, pp. 173–180.

Leotta, M., Ricca, F., Ribaudo, M., Reggio, G., Astesiano, E., Vernazza, T., 2012. SOA adoption in the Italian industry. In: Proceedings of the 2012 Intl. Conference on Software Engineering, IEEE Press, pp. 1441–1442.

Li, Z., O'Brien, L., Zhang, H., 2012. Circumstantial-evidence-based effort judgement for web service composition-based SOA implementations. Int. J. Space-based Situat. Comput. 2 (1), 31–44.

Liu, Y., Wang, Q., Zhuang, M., Zhu, Y., 2008. Reengineering legacy systems with restful web service. In: 32nd Annual IEEE Intl. Computer Software and Applications. COMPSAC'08 IEEE, pp. 785–790.

Madani, N., Guerrouj, L., Di Penta, M., Gueheneuc, Y., Antoniol, G., 2010. Recognizing words from source code identifiers using speech recognition techniques. In: 2010 14th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, pp. 68–77.

Maleshkova, M., Pedrinaci, C., Domingue, J., 2010. Investigating web apis on the world wide web. In: 2010 IEEE 8th European Conference on Web Services (ECOWS), IEEE, pp. 107–114.

Manning, C.D., Schutze, H., 1999. Foundations of Statistical Natural Language Processing, vol. 999. MIT Press, Boston.

Markey, C.G., Philip, 2013. A performance analysis of WS-* (soap) & RESTful Web services for implementing service and resource orientated architectures. In: The 12th Information Technology and Telecommunications (IT&T) Conference, Athlone IT.

Miller, G.A., 1995. Wordnet: a lexical database for English. Commun. ACM 38 (11), 39–41.

U.N. National Cancer Institute, 2009. Service and Capability Naming Standards Document. https://wiki.nci.nih.gov/display/SAIF/CBIIT+SAIF+Wiki

OASIS, 2010. The oasis ws-i. http://www.oasis-ws-i.org/

Pautasso, C., Wilde, E., 2009. Why is the web loosely coupled? A multi-faceted metric for service design. In: Proceedings of the 18th international conference on World wide web, ACM, pp. 911–920.

Pautasso, C., Zimmermann, O., Leymann, F., 2008. Restful web services vs. big'web services: making the right architectural decision. In: Proceedings of the 17th international conference on World Wide Web, ACM, pp. 805–814.

Perrone, M.P., Cooper, L.N., 1992. When Networks Disagree: Ensemble Methods for Hybrid Neural Networks, Tech. Rep., DTIC Document.

Rensink, A., 2004. The groove simulator: a tool for state space generation. In: Applications of Graph Transformations with Industrial Relevance, Springer, pp. 479–485.

Renzel, D., Schlebusch, P., Klamma, R., 2012. Today's top "restful" services and why they are not restful. In: Web Information Systems Engineering – WISE 2012, Springer, pp. 354–367.

Rodriguez, J.M., Crasso, M., Mateos, C., Zunino, A., Campo, M., 2011. The easysoc project: a rich catalog of best practices for developing web service applications. CLEI Electron. J. 14 (3), 2–2.

Rokach, L., 2010. Pattern Classification Using Ensemble Methods, vol.75. World Scientific, Singapore.

Salton, G., Wong, A., Yang, C.-S., 1975. A vector space model for automatic indexing. Commun. ACM 18 (11), 613–620.

Schreier, S., 2011. Modeling restful applications. In: Proceedings of the Second International Workshop on RESTful Design, ACM, pp. 15–21.

Selonen, P., 2011. From requirements to a restful web service: engineering content oriented web services with rest. In: REST: From Research to Practice. Springer, New York, pp. 259–278.

Sheth, A.P., Gomadam, K., Lathem, J., 2007. Sa-rest: semantically interoperable and easier-to-use services and mashups. IEEE Internet Computing. 11 (6), 91–94.

Strauch, J., Schreier, S., 2012. Restify: from RPCs to RESTful HTTP design. In: Proceedings of the Third International Workshop on RESTful Design, ACM, pp. 11–18.

Tarjan, R., 1972. Depth-first search and linear graph algorithms. SIAM J. Comput. 1 (2), 146–160.

Upadhyaya, B., Zou, Y., Xiao, H., Ng, J., Lau, A., 2011. Migration of soap-based services to restful services. In: 2011 13th IEEE International Symposium on Web Systems Evolution (WSE), IEEE, pp. 105–114.

Vinoski, S., 2002. Putting the "web" into web services. web services interaction models. 2, Internet Computing. IEEE 6 (4), 90–92.

Vinoski, S., 2008. Serendipitous reuse, Internet computing. IEEE 12 (1), 84–87.

Voutilainen, A., 2003. Part-of-speech Tagging, The Oxford Handbook of Computational Linguistics., pp. 219–232.

W3C, 2002. Web Services Activity. http://www.w3.org/2002/ws/

Michael Athanasopoulos graduated from the School of Electrical and Computer Engineering at National Technical University of Athens, Greece, where he is currently a Ph.D candidate. Michael has worked as a software engineer in the banking sector for several years and he has significant experience with enterprise service systems. His research interests include service-oriented computing, software architectural styles and architectural evolution. Currently, he is working on his dissertation on architectural adaptation of SOAs to resource-oriented architectures. Michael held a Ph.D. Fellowship Student at Center of Advanced Studies, IBM Canada.

Kostas Kontogiannis holds a B.Sc. from the University of Patras, Greece, a M.Sc. from the Katholieke Universiteit Leuven, Belgium and a Ph.D. from McGill University, Canada. He is an Associate Professor at the School of Electrical and Computer Engineering at NTUA, Greece. Previously, he was an Associate Professor at the Department of Electrical and Computer Engineering at the University of Waterloo, Canada. Kostas is a Faculty Fellow at the IBM CAS Research. His current research interests focus on the design and development of tools for software reengineering software transformations, dynamic analysis, and legacy software migration to service computing platforms.