2010 17th Working Conference on Reverse Engineering

Identification of REST-like Resources from Legacy Service Descriptions

Michael Athanasopoulos Kostas Kontogiannis
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
athanm@softlab.ntua.gr

***Abstract*— Service-oriented systems mainly follow two principles for accessing data and invoking back end applications: Remote Procedure Calls and Message-Orientation. However, a number of researchers and practitioners have criticized these paradigms as too complex and rigid. Instead, Representational State Transfer (REST) architectural style has lately gained significant attention as an alternative means for accessing services and data. RESTful HTTP systems depend on Uniform Resource Identifiers (URIs) to uniquely identify and denote data and services as "resources". In this paper, we discuss a technique to analyze the descriptions of legacy data and services in order first, to model their roles and relationships and second, to use the discovered dependencies for extracting Unique Resource Identifiers and the available HTTP methods, so that these legacy service elements and data can be accessed using lightweight requests.**

Keywords- Service-Oriented Systems, REST, Migration, Software Architecture

I. INTRODUCTION

In today's corporate environments large software applications are built as a collection of components that provide and consume services and data, utilizing a variety of diverse service description, communication and invocation protocols. However, the technical complexity and structural diversity of these protocols have been identified as primary stumbling blocks for the ease of development and widespread adoption of such service-oriented systems. In order to overcome these shortcomings a lot of interest is currently being concentrated on software engineering methods and tools related to the development of applications that conform to the REST [1] architectural style. RESTful services allow for significant simplifications and flexibility regards to development, deployment and invocation of web services. Also, REST as the architectural style of the Web, when used in application integration may bring improvements: in scalability through statelessness, in performance through caching, in long-term compatibility and in evolveability through content types that can either evolve independently or new ones can be added without dropping or reducing support for existing ones. Being able to expose existing legacy functionality through RESTful interfaces would allow for significant reuse benefits of well-tested value-proven systems into a variety of contexts that arise from current business needs. In this context, in order to achieve the exposure of services and data which were previously hidden behind Web Service endpoints to the global namespace of Web, an important step forward is to introduce techniques for modeling their relationships and analyzing their dependencies with the intention first of being able to identify them via Uniform Resource Identifiers and second identifying their available manipulation actions (i.e. POST, PUT, GET, DELETE).

In this paper, we propose a model-driven engineering approach to allow for the identification of REST-like resources using as input legacy service descriptions. More specifically, standard legacy service signatures described in an Interface Description Language (IDL) such as WSDL are represented in a MOF compliant model referred to as the *Signature Model* and legacy data are represented using a corresponding *Data Model*. Consequently, model transformation techniques are applied to create a service and data dependency graph that captures the semantic and structural dependencies among these elements. Such a dependency graph is then refined in order to disambiguate probable dependency conflicts and analyzed to generate equivalence classes of Uniform Resource Identifiers (URIs) that can be used to access the legacy data and services in a RESTful manner.

Accessing legacy services and data in a RESTful manner has significant advantages. First, it allows for new applications to be built by utilizing widgets in a mashup fashion. Second, it provides a framework whereby Internet users can compose their own Internet space which is defined as a collection of resources that can be used to feed, filter, compose, disseminate and reference information, data, and services to users according their profile, context, and mode of operation. Third, the convergence of a RESTful and SOA type of programming model creates opportunities for new service architectures and SOA programming models, where service components can be accessed through multiple bindings (e.g. both Atom and SOAP), according to the context they are invoked and used on.

This paper is organized as follows. Section 2 is discussing related work. Section 3 is presenting the modeling of legacy elements. Section 4 is presenting the dependency analysis process over the legacy service and data elements and the formation of Uniform Resource Identifiers for

1095-1350/10 $26.00 © 2010 IEEE
DOI 10.1109/WCRE.2010.31

accessing and manipulating these elements. Section 5 outlines results obtained from the analysis of the Open Travel Alliance (OTA) schema and services standard. Section 6 presents a discussion of the emerging issues in this area and concludes the paper.

II. Related Work

An approach on formalizing RESTful web service descriptions so that automated composition techniques can be performed is discussed in [2]. Authors introduce a classification of three types of RESTful web services namely: a) Type I (Resource Set Service) which refer to collections of domain resources (e.g. a set of *Orders*), b) Type II (Individual Resource Service) which refer to individual domain resources and can be used to denote instance level resources (e.g. an actual *Order* that can be reached through some ID), and c) Type III (Transitional Service) which refer to services that participate in some form of transition or transformation of other resources' states.

The significant diversifications in the web services domain as regards to (design, development and description) technologies, protocols and architectural paradigms, highlight the need of defining and using a more abstract mechanism of service description and processing which would provide invariant semantic models that are "immune" to rapid technological swifts. Motivated by that, researchers in [3] introduce a service abstraction model which allows for a more lightweight way to describe web services and is more flexible with handling technological diversifications.

Finally, in [4] the authors present a model-driven process for gradually migrating from a set of functional service requirements to a resource-centric design of web services while employing model transformations.

III. Modeling Legacy Components

The first step to exposing existing legacy functionality as a collection of RESTful resources, is to denote service descriptions of legacy components into a common model based on service signatures, domain data model schemas, and a classification UML profiling process over the type of operations and the types of domain data elements.

A. Signature Model

The Signature Model is a MOF model, depicted as a class diagram with the extension of a profile extracted from the classification of service's operations and their parameters. The classification process for each element of the Signature Model is discussed in the following section. The Signature Model stems from the following steps that are repeated for every operation in a service description specification:

- For each element of the signature (operation, input and output types) a class is created.
- Operations are connected to their input and output parameters with associations annotated with the role of the parameter as IN, OUT, or IN/OUT parameters.
- The classification profile that is discussed in the following section is then applied to the model and stereotypes are assigned to the model classes.
- Finally, for every parameter stereotyped as *Container Element* that is not part of a respective *Container*, a class stereotyped as *Container* is created as well as a containment association between them.

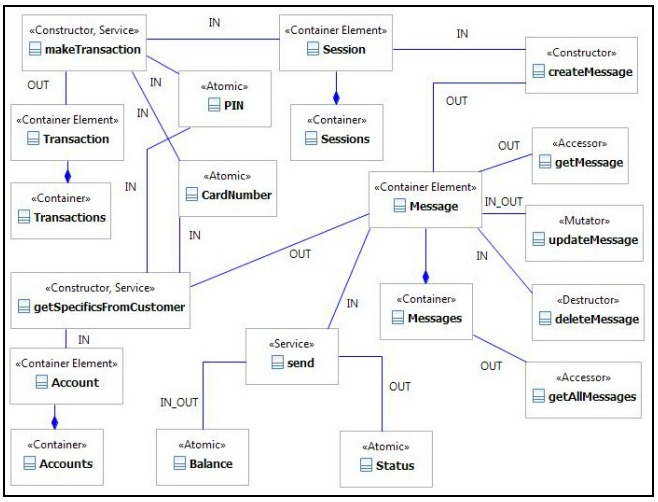

Figure 1. ATM operations' Signature Model

As an example consider the signatures of several operations derived from an ATM simulation system [6]. In this example, an ATM operation that was used to generate part of the Signature model depicted in Figure 1 is <*makeTransaction*, IN:*PIN, CardNumber, Session*, OUT:*Transaction*>. After applying the mapping rules described above for eight operations, the complete Signature Model is presented in Figure 1.

B. Signature Model Element Classification

In the proposed approach, the classification process is based on heuristics which mostly depend on how rich the descriptions of the services and of the domain model are. We have been experimenting with several heuristics but at least for now, we regard the process as semi-automatic, meaning that human involvement may be demanded in order to guide the classification steps and review or adapt the results.

As mentioned above, the classification process is used to define a profile which will be later applied on the MOF classes of the Signature Model to yield the final profiled Signature Model. The classification types we consider for operations are *Constructor*, *Destructor*, *Accessor*, *Mutator*, *Query*, *Investigator* and, *Service*. Similarly, the classification types we consider for parameters are *Container Element*, *Container* and, *Atomic/Transient Data*.

Through the classification process, parameters are categorized into exactly one class and operations are

categorized into at least one class. Operations that are mapped into exactly one class are called *pure* (e.g. an operation classified only as a *Constructor*), while operations mapped into two or more classes are called *complex* (e.g. an operation classified as a *Constructor* and also as a *Service*). As implied by the above, an operation may have one or more stereotypes assigned to it and a parameter must have exactly one.

Finally, in the case of *complex Constructor* operations that have more than one output parameters, their associations with these parameters are further characterized either as *creation* (when the parameter is constructed and returned by the operation) or, *retrieval* (when the parameter value is retrieved or computed by the operation).

IV. Identification of Service and Data Resources

Once the Signature Model is extracted from the available legacy components' descriptions and the classification process is applied, we proceed by performing an analysis over the elements contained in the Signature Model in order to investigate potential resources and resource hierarchies that will guide the Resource identification process, forming the URIs and the appropriate HTTP methods to be used with. The identification process has three steps, a) the selection of potential resources; b) the creation of signature dependency graphs and; c) the identification of a resource model for the formation of URIs and their corresponding actions.

A. Potential Resources

Potential Resources constitute a subset of the Signature Model's elements, upon which the dependency analysis is performed. Potential Resources are extracted employing heuristics over Signature Model's annotations and structural properties. In our current approach we consider as Potential Resources all the data elements stereotyped as *Container* and *Container Element* and, all the operations stereotyped as *Service*. Other heuristics may reflect domain or model specific constraints and assumptions, design decisions, as well as business-oriented rules or conventions.

B. Construction of Signature Dependency Graphs

The first step in the proposed dependency analysis process is constructing and refining a directed graph called *Signature Dependency Graph (SiDG)* which will be transformed for the purposes of URI formation into a directed acyclic graph (DAG). The SiDG is constructed automatically taking into account only a subset of the Signature Model's elements and their associations. Using this information the SiDG is formed through the following mapping rules:

- Single *Container* signature model classes as well as pairs of *Container* and *Container Element* classes are mapped to SiDG vertices.
- Service operations are also mapped to SiDG vertices.
- For every *pure Constructor*, signature model class directed edges are created emanating from every vertex that corresponds to an OUT parameter

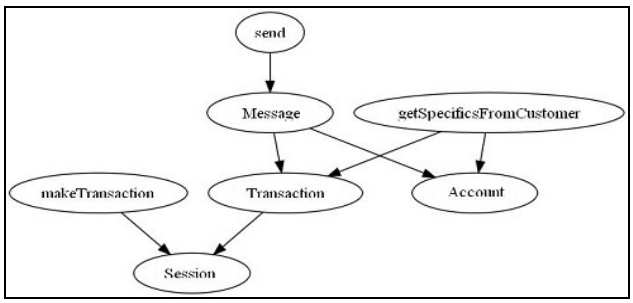

Figure 2. SiDG example

associated with the *Constructor* and terminating to all the vertices that correspond to IN parameters associated with this *Constructor*. When the parameters are stereotyped as *Container*, the endpoints of the created edges are labeled with a star denoting multiplicity.

- For every *complex Constructor*, edges are created in the same fashion as above with the difference that only OUT associations that are labeled as *construction*, are considered.
- For every *Service* signature model class, directed edges are created emanating from the respective vertex and terminating to all the vertices that correspond to the IN parameters associated with the operation, following the same rule as above as regards edge endpoints.

Figure 2 illustrates the SiDG that is constructed from the Signature Model depicted in Figure 1.

C. Vertex identification / contraction

After constructing the SiDG, a refinement process may have to take place in which all the strongly connected subgraphs are contracted into higher level vertices. Finally, a directed acyclic graph which is a condensation of the SiDG (referred to as SiDGc) is constructed and will be used to form the URIs (or the URI equivalence classes). The contracted vertices are considered as composite Potential Resources and are identified and exposed through URIs in the same way as the rest of the resources. We use Gabow's algorithm [5] (also known as Cheriyan-Mehlhorn algorithm) for the identification of strongly connected components. Figure 3 shows a sample SiDG on the left and its condensation on the right. Potential Resources E, F, and G are contracted to EFG and the resulting graph is directed and acyclic.

D. Service Resource Model

Since a SiGDc is a DAG, the vertices (i.e. simple and composite Potential Resources) are partially ordered. In case that there is a Hamiltonian path in SiGDc the resource hierarchy for the whole application is intuitive and the URI formation for each potential resource is reduced to just traversing the path from that resource to the "sink" of the graph and appending the resource names in the reverse order. However, usually the case is that there is more than one path from each Potential Resource to "sink" resources and this is

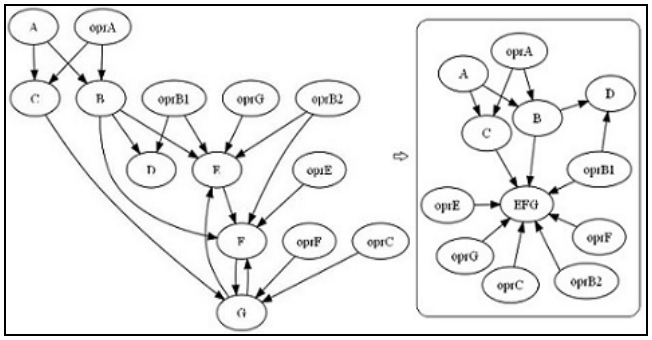

Figure 3. Vertex Contraction example.

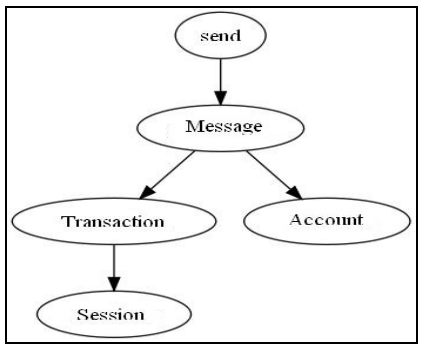

Figure 4. *send* SiDG subgraph.

when *URI equivalence classes* are introduced. A URI equivalence class is represented with the "||" operator which denotes all permutations between the associated resources. For example, the http://.../A||B URI for the resources A and B, will be an equivalence class for http://.../A/B or http://.../B/A URIs.

E. Transitional Resources

As mentioned above, Potential Resources include operations stereotyped as *Service*. These resources would be classified as Type III category of services presented in [2] and we refer to as *Transitional Resources*. Such a resource is accessed through a POST request to its URI. In order to demonstrate how the URI is formed we take as an example the ATM *send* operation. Figure 2 illustrates the SiDG corresponding to the ATM example which is already a DAG, meaning that we may skip the vertex contraction step since the resulting SiDGc will be exactly the same.

Consequently, the URI equivalence class for the send operation is formed by the subgraph depicted in Figure 4 and is translated into textual form as:

```
http://.../((sessions/{sessionid}/transaction
s/{transactionid})||(accounts/{accountid}))/m
essages/{messageid}/send
```

where "||" stands for *parallel* operator. Also, when an edge endpoint is labeled with a star (*) only the collection resource name corresponding to that vertex appears on the URI pattern. Topological sorting of the above example leads to the following two URIs:

```
http://.../sessions/{sessionid}/transactions/
{transactionid}/accounts/{accountid}/messages
/{messageid}/send
```

and,

```
http://.../accounts/{accountid}/sessions/{ses
sionid}/transactions/{transactionid}/messages
/{messageid}/send
```

F. Data Resources

Data Resources contained in the SiDG (or SiDGc if necessary) can be also identified through URI equivalence classes in exactly the same way that was described for Transitional Resources. For example, the SiDG subgraph for the *Message* vertex is the same with the one in Figure 4 after removing the *send* operation vertex and the edge emanating from it. Consequently, individual *Message* resources are identified through the following URI pattern which is also the URI pattern for the respective resource collection after removing the last resource ID segment.

```
http://.../((sessions/{sessionid}/transaction
s/{transactionid})||(accounts/{accountid}))/m
essages/{messageid}
```

The ways in which these resources can be manipulated (i.e. the available HTTP methods) is of course a matter of the available functionality, and in particular of the *pure* legacy operations. In this context, we introduce five rules to map legacy service operation invocations "genuinely" to RESTful HTTP requests over identified data resources.

Creation. An operation stereotyped as *Constructor* that creates a *Container Element* is mapped to a POST request to the URI of the corresponding *Container*.

Retrieval. An operation stereotyped as *Accessor* that retrieves the contents of a *Container* or a *Container Element* is mapped to a GET request to the corresponding URI.

Modification. An operation stereotyped as *Mutator* that modifies the contents of a *Container* or a *Container Element* is mapped to a PUT request to the corresponding URI.

Removal. An operation stereotyped as *Destructor* that removes a *Container* or a *Container Element* is mapped to a DELETE request to its corresponding URI.

View. An operation stereotyped as *Query* that returns data which is a part or a view of a *Container* or a *Container Element*, based on query parameters is mapped to a GET request to the corresponding URI followed by the query parameters as a sequence of property-value pairs separated by a delimiter character.

V. EXAMPLE: OPENTRAVEL AIR SERVICES

As an application example, we consider the schema from Open Travel Alliance (OTA) [7]. For brevity, we constrained our analysis to air traveling related messages. The resulting Signature Model (a segment of which is depicted in Figure 5) contains the following operations: *OTAAirBook*, *OTAAirBookModify*, *OTAAirRules*, *OTAAirCheckIn*,

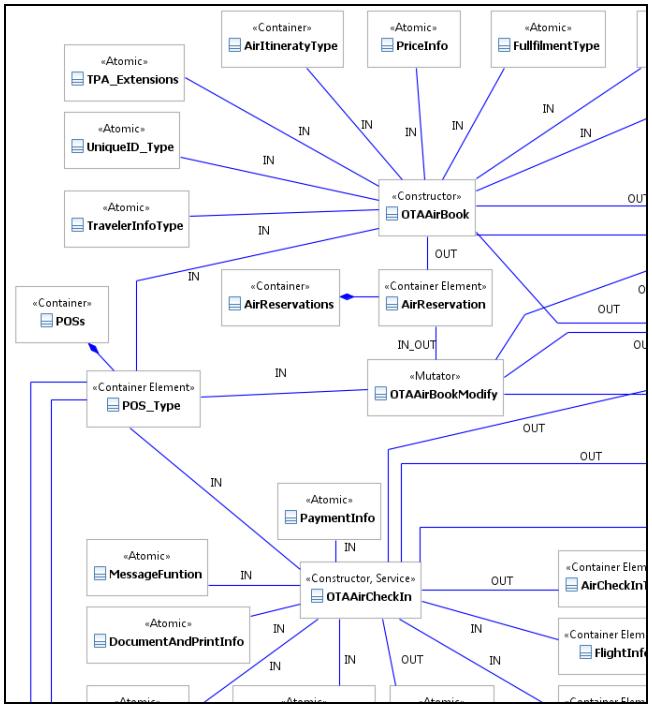

Figure 5. OTA Air Services Signature Model segment.

OTAAirAvail. The operations *OTAAirBook* and *OTAAirBookModify* are *pure* operations stereotyped as *Constructor* and as *Mutator* respectively and they are both related to the resource *AirReservation* which is stereotyped as *Container Element*. *OTAAirRules* is stereotyped as *Query* and returns a collection of *FareRuleResponseInfo* resources. The rest of the operations are *complex*. Specifically *OTAAirCheckIn* is a *complex Constructor* of *AirCheckInType* resources and *OTAAirAvail* is stereotyped both as *Service* and as *Query*. Once the Signature Model is constructed and its elements are stereotyped we proceed by executing the Signature Dependency Graph extraction rules. The SiDG generated by the Signature Model is presented in Figure 6 while some of the resulting URIs include:

`http://.../(POS/{POSID})||(AirItinerary/{AirItineraryID})/AirReservations/` (POST operation)

`http://.../(POS/{POSID})||(AirItinerary/{AirItineraryID})/AirReservations/{AirReservation}/` (PUT Operation)

`http://.../FareRuleResponseInfo/?RuleReqInfo={parameter values}` (GET Operation)

`http://.../(POS/{POSID})||(PassengerInfo/{PassengerInfoID})||(FlightInfo/{FlightInfoID})/OTAAirCheckIn` (POST operation) and,

`http://.../(POS/{POSID})||OriginalDestinationInformation/OTAAirAvail` (POST operation)

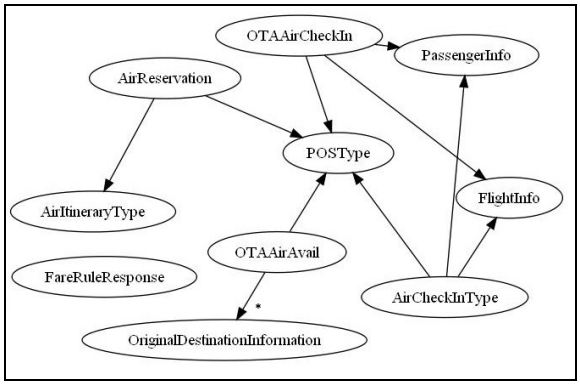

Figure 6. OTA Air Services SiDG.

VI. CONCLUSIONS AND DISCUSSION

In this paper, we presented a model-driven approach in identifying REST-like Resources from legacy service descriptions. Using the information contained in the descriptions of the available functionality (in the form of WSDL or Message schema specifications) we proposed a way to model service operations signatures into a MOF model called Signature Model. The Signature Model captures structural and semantic information about its elements and their associations. This model is then used to extract directed graphs depicting dependencies between a subset of signature model elements characterized as potential resources. Based on that dependency analysis, URI equivalence classes are extracted for every resource and topological sorting is proposed as a way of forming unique identifiers. Furthermore, we introduced a set of rules that can be used to map existing operations to truly RESTful HTTP requests when specific patterns are present in the Signature Model. Finally, issues such as the usage of appropriate MIME types that would carry "what goes into the HTTP interactions message payloads", how these mappings are done, and how this information could be used to improve the URI formation process, is subject of future work.

REFERENCES

[1] R. Fielding. "Architectural Styles and The Design of Network-based Software Architectures". PhD thesis, University of California, Irvine (2000)

[2] H. Zhao, P. Doshi, "Towards Automated RESTful Web Service Composition," IEEE International Conference on Web Services, 2009, pp.189-196

[3] Li, L. and Chou, W. 2009. "Infoset for Service Abstraction and Lightweight Message Processing". In Proc. of the 2009 IEEE international Conference on Web Services (July 06 - 10, 2009). pp. 703-710.

[4] M. Laitkorpi, P. Selonen, P., and T. Systa. "Towards a Model-Driven Process for Designing ReSTful Web Services". In Proc. of the 2009 IEEE international Conference on Web Services, pp. 173-180.

[5] Gabow, H.N. (2003), "Searching (Ch 10.1)", in Gross, J. L.; Yellen, J., Discrete Math. and its Applications: Handbook of Graph Theory, 25, CRC Press, pp. 953–984 .

[6] ATM Simulation, http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/

[7] OpenTravel Alliance, http://www.opentravel.org/