Modeling Behavioral RESTful Web Service Interfaces in UML

Ivan Porres and Irum Rauf
Åbo Akademi University, Dept. of Information Technologies, Turku, Finland
ivan.porres@abo.fi, irum.rauf@abo.fi

ABSTRACT
A web service interface contains information about the names of the operations that can be invoked on the service and the input and output parameters of these operations. The Web Application Description Language (WADL) is a language to describe the interface of a web service that follows the Representation State Transfer (REST) architectural style. Currently, WADL descriptions do not describe the behavioral semantics of the operations neither ensure that the published interfaces follow the REST style, that is they are RESTful. In this paper, we present an approach to model the structural and behavioral interface of a RESTful web service using UML class and UML protocol diagrams. These models lead to RESTful interfaces that describe the behavior of operations in terms of preconditions and postconditions. The contracts can then be published in an extended version of the WADL language and used for documentation, stub generation, testing and monitoring purposes.

Keywords
REST, contract, UML, protocol state machine, WADL, behavioral interface

1. INTRODUCTION
A web service provides a programmatic interface to a computer system offered over a network using standard web protocols. Web services are consumed by other programs and services. This is unlike the document web or an interactive web application that is designed for human users who search for a particular information, receive it in a format that they understand and then use it manually as input to another software or machine for further processing.

The Representational State Transfer (REST) architectural style has become a popular approach to design web services. It emphasizes on generality of interfaces and scalability of component interactions. A web service that follows the REST style is often called a RESTful web service.

Many RESTful web services present simple interfaces to create, retrieve, update and delete information from a database (also called CRUD interfaces). However, REST is not limited to simple CRUD interfaces. It is possible to create web services with a complex application state that still follow the REST architectural style. In these cases, it is important to create and publish behavioral service interfaces so other developers can understand how to use a service correctly.

The interface of a web service advertises the operations that can be invoked on it. A web service developer looking for a particular service finds the service over the web and integrates it with other services by invoking the advertised operations and providing it the required parameters. These operations may require a certain order of invocation or there may be special conditions under which they can be invoked. This information, together with the expected effect of an operation forms part of the behavioral interface of a service. The behavioral interface can also be used to test a service implementation and for service discovery. More advanced scenarios, such as automatic service discovery and service repositories rely in formal descriptions of services.

There is a proposal for a description language for RESTful web services named the Web Application Description Language (WADL). It is used for publishing RESTful web service interfaces and provides machine-processable description of the interface without cluttering it with too many details [9]. Currently, RESTful architectural style and WADL are being widely adopted in the web and have numerous users, including enterprises such as Google, Yahoo, Amazon and Flicker.

The information about the interface of a web service in WADL is syntactic and does not state anything about its semantics, how a service should be invoked and behave. Also, WADL is not actually restricted to RESTful services and can be used to describe services that do not follow this architectural style completely.

In this paper, we study how to use UML to model and describe behavioral interfaces for web services that follow the REST style. The objective of the work is twofold. First, to provide a way to describe behavioral services interfaces that specifies how to use a web service correctly and what are the expected results of using a web service. Second, to provide a modeling approach that ensure that the designed interface follows the REST style.

To demonstrate our approach, we use as example an imaginary hotel room booking (HRB) service. The service allows a client to book a room, pay for the reservation, and cancel it. It is a simplified pedagogical example, but it shows...
how to design a REST interface for a service with a complex application state.

The paper is organized as follows. Section 2 gives an overview of the approach. The conceptual resource model and the behavioral model are presented in Section 3 and 4, respectively. Section 5 presents the generation of contracts from models and behaviorally enriched WADL. Section 6 shows how these models lead to RESTful interfaces. Section 7 discusses the application of these behavioral interfaces. In section 8, related work is presented and Section 9 provides conclusion of the paper along with the future work.

2.Overview

In this article we propose an approach that creates RESTful web services by construction. We use UML class diagrams and UML protocol state machines with state invariants to model a REST web service. These models can also be used to generate a contract as its behavioral interface.

The REST architectural style is defined by four attributes. In the context of a web service these attributes are:

- **Addressability**: The REST style requires that any important piece of information related to a service should be exposed as a resource and each resource should be addressable via a URI.
- **Connectedness**: This requires that resource representation contains links to other resources.
- **Uniform Interface**: All resources are manipulated using the standard HTTP methods. The HTTP GET, POST, PUT and DELETE are used to retrieve information from a resource or change its state.
- **Statelessness**: There is no hidden session or state information. Besides, the effects of the POST, PUT and DELETE operations should be observable in the affected resources.

Any RESTful web service should comply with these four attributes. Therefore the attributes become requirements over the design of the web service interface.

The structural and behavioral UML models proposed in this article for a web service comply with these requirements and lead to RESTful interfaces.

UML is a standard modeling notation that provides representation of the system in an abstract manner from different perspectives. It also serves as part of the specification document.

The starting point of our approach is an informal web service specification in natural language. This specification is used to create a conceptual resource model and a behavioral model of the web service. The conceptual resource model is represented diagrammatically using a UML class diagram. It represents the resources involved in a service and their connections and it tackles the addressability and connectedness requirements of the REST style. The behavioral model of the service is represented using a UML protocol state machine with state invariant. A protocol state machine contains a number of states and transitions. Each transition is triggered by a method and each state should have an invariant. By constraining the allowed transition triggers to the standard HTTP method we comply with the uniform interface requirement and by defining the state invariants using addressable resources we comply with the statelessness requirement.

A contract specifies the pre- and postconditions for the methods of a class. The role of contracts as behavioral interfaces has been investigated for software classes [15, 7, 5] and also in the domain of web services [10, 8]. A contract binds the user of the service to pose a valid request and constrains its provider to provide the correct behavior. We show how the pre-and postcondition of each service request can be generated from the proposed UML models and how these pre- and post conditions can provide behaviorally enriched WADL interface.

Our approach can generate implementation stubs and also extract information from behavioral model of a web service and assert it in syntactic interface of a web service. The extended WADL descriptions provide behavioral interface specifications in a machine-processable format.

The developed models and contracts can be used to implement the actual web service using a design by contract approach [15]. They can also be used to test an existing implementation by generating test cases automatically from the contracts [13, 1, 15]. Finally, users of a web service can use the models and contracts as detailed documentation on how to use a service correctly.

The service can be described using the UML language or using the WADL extended with preconditions and postconditions. UML models facilitate the design process as they can be read and understood by human developers. The WADL descriptions are in a machine readable format that can be processed by machines for extracting the required information.

In the next section we present in detail how a conceptual model is developed for a RESTful web service.

3. Conceptual Resource Model

The concept of a resource is central to Resource Oriented Architecture. Any important information in a service interface is exposed as a resource. A resource is something that can be referred to and can have an address. Resources in a resource oriented architecture are analogous to objects in object-oriented paradigm [18] or entities in the entity-relationship model.

However, unlike objects and classes in OO paradigm, the resources in resource oriented paradigm are designed with the mind set to expose relevant information for manipulation instead of hiding it [19]. Also, in a RESTful interface, resources do not have different access methods, instead the standard HTTP methods are used. Our approach uses four HTTP methods i.e, GET, PUT, POST and DELETE, for retrieving and updating data in a resource.

We use UML class diagram to represent the structure and connections of the resources in a RESTful web service. A UML class diagram represents mainly classes and their associations. An association defines a relationship between two classes by which one class knows about the other class [20].

The complexity of a service can be reduced by increasing the number of resources. This results in decoupling of information. A resource can also be a collection resource that contains a group of other resources. Figure 1 shows a conceptual model of the HRB RESTful service. We have broken our HRB service into two collection resources (bookings and
rooms) and five other resources (booking, room, payment, pconfirmation and cancel).

3.1 Mapping Class Diagrams to Resources

The elements of a conceptual model are mapped to the resources of a RESTful web service as follows:

- Each class in the UML diagram represents a resource.
- Classes with the << collection >> stereotype represent collection resources.
- The UML class attributes are the data contained in the resource. This data should appear in the resource representation, i.e., an XML document or a JSON serialized object.
- The UML associations represent the connections between resources.
- The role names on association ends map to the relative navigation path from one resource to another.
- The association multiplicity refers to the number of resources that can be related to a resource on the other end of the association.

Figure 1 shows representation of resources in the HRB service. For example, room resource contains three attributes i.e. rid, rType and floor. Room ID(rid) and floor(floor) are integer values and Room Type(rType) is a string value. Attributes are modeled as a public attribute as the representation of a resource is available for manipulation.

3.2 Collection Resources and Addressability

In Figure 1, bookings and rooms represent collection resources with the stereotype collection and are linked to child resources, booking and room, respectively. A collection resource has a cardinality of more than 1 on the association end of a child resource. A GET method on a collection resource returns a list of all the child resources it contains. For example, a GET method on bookings will give a list of all the booking resources that it contains.

Collection resources are also used as the starting point of the paths to address each resource. Starting from a collection resource, we can access other resources by navigating the successive associations. Paths visiting the same association more than once are not valid. In our example, the following paths are valid.

The REST style requires that all resources should be addressable. In our context this requirement is fulfilled if each resource can be reached from at least one collection resource by navigating one or more associations.

3.3 Methods

A UML class diagram allows us to define a number of operations for each class. Since a RESTful web service provides uniform interface for all resources, most classes would only have from one to four methods names GET, POST, PUT and DELETE. We do not consider necessary to add this information in the conceptual model.

4. BEHAVIORAL MODEL

The purpose of the behavioral model is to describe the behavioral interface specifications of a RESTful web service. It shows the sequence under which operations should be invoked, the conditions under which they can be invoked and the expected results.

Since we are describing RESTful web interfaces, the only allowed operations are GET, POST, PUT and DELETE on resources.

The GET method retrieves a representation of a resource and it should not have side effects. Due to the addressability requirement, it should be always possible to invoke a GET method over a resource. For example, GET(/bookings/{bid}/payment) and GET(/bookings/{bid}/rooms/{rid}) are HTTP GET methods on the resources payment and room, respectively. Whenever a GET method is called on a resource, it gives the representation of resource as a response. In practice, the access to resources may be restricted by an authentication and access control mechanism.

The POST, PUT and DELETE methods can have side effects. In our hotel booking example, one of the operations of the service is to pay a booking. This is achieved by a HTTP POST request over a payment resource. However, a payment can only be accepted if it is connected to a room booking and a booking can only be paid once. Also, a booking can be canceled, but not while the payment is being processed by a financial service. We need to define all these conditions in the behavioral interface of the service.

We propose to use a UML protocol state machine with state invariants to describe the allowed operations in a web service. We consider that a UML protocol state machine is suitable for representing the behavior of a web service as it provides interface specifications with information on conditions under which methods can be invoked and the expected output from them.

A UML protocol state machine contains mainly states and transitions. We require that each state has a state invariant that is defined as a boolean expression. We then say that a state is active if and only if its state invariant evaluates to true. A state may contain other states and is called a composite state. In such a case, the actual state invariant of the contained state is given by the conjunction of the state invariant specific for the contained state and the state invariants of all the states that contain it. These state invariants within a composite state should be mutually exclusive. That is, only one state within a region of a composite state can be active at a time.

A transition is an arc from one or more source state(s) to one or more target state(s) labeled with a method name and a guard. If the source states are active, the guard is true and the method is invoked, then the transition may be fired and as a consequence the target state(s) become active. When no guard is shown in the transition it is assumed to be true.

In our behavioral model, the transition triggers can only be defined as POST, PUT or DELETE operations over resources described in the conceptual model. The guards and the state invariants can be defined only using information from the resources and request parameters.

When we invoke an HTTP GET method on a resource, it returns its representation along with the HTTP response code. This response code tells whether the request went well or bad. If the HTTP response code is 200, this means that
the request was successful and the referred resource exists. Otherwise, if the response code is 404, this implies that URI
could not be mapped to any resource and the referred re-
source does not exist.

We manipulate this information exhibited by the response
codes and representation of resources by comparing them
with the expected behavior. We use a boolean function
\text{OK}(r)\) to express that the response code of HTTP GET
method on a resource \(r\) is 200. Similarly, the boolean func-
tion \text{NOT_FOUND}(r)\) is true when the response code of
HTTP GET method on resource \(r\) is 404. These boolean
functions on the resources along with the attributes that
represent a resource are used to define a state invariant in
our RESTful behavioral model.

For example, consider the state invariant for the state
\text{reserved_not_paid} in Figure 2. \text{NOT_FOUND}(p) checks the re-
sponse code for the HTTP GET method on the resource \text{pay-
ment}. It evaluates to true if response code of GET method
on \(p\) for a particular booking ID(\{bid\}) is 404. For the HRB
service to be in state \text{reserved_not_paid}, the state invariant
of this simple state is conjuncted with the state invariants
of all the states that contain it.

Our behavioral model shows different states of a RESTful
web service and gives information on what HTTP methods
on a particular resource can be invoked from a certain state.
According to Figure 2, the protocol state machine of HRB
service is initiated by the HTTP POST method on the \text{book-
ings} resource. Whenever a PUT, POST or DELETE method
is called on a resource it changes the state of the application
in the behavioral model. A booking can be canceled from
composite states \text{reserve_and_pay} and \text{confirmation_info} and
a booking can be deleted only if it is canceled. A booking
cannot be canceled if it is waiting for the payment confirma-
tion from a third-party service.

A guard condition on the transition specifies the condi-
tion required to invoke an HTTP method on a resource. For
example, consider guard \{pc.confirm==true\} for the method
\text{POST}(pc\text{confirmation}) in Figure 2, where \(pc\) refers to the rel-
ative navigation path to resource \text{pcconfirmation}. This guard
specifies that the POST method on \text{pcconfirmation} can be
invoked if the resource representation of \text{pcconfirmation}\) con-
tains true value for the \text{confirm} attribute.

4.1 Stateless State Machines?

Using a state machines to model a stateless interface may
seem an oxymoron. In the context of a RESTful service,
statelessness is interpreted as the absence of hidden informa-
tion kept by the service between different service requests.
In that sense, a RESTful web service should exhibit a state-
less protocol. Also, there is no sense of session or sequence
of request in a true RESTful service.

On the other hand, state machines have a notion of ac-
tive state configuration, that is, what states are active at a
certain point of time. If an implementation of an interface
described using a state machine would have to keep the ac-
tive state configuration between different requests, then this
would break the statelessness requirement of the RESTful
service.

However, our approach does not actually require that a
service implementation keeps any additional protocol state.
In our approach a state is active if its invariant evaluates to
true, but the invariants are defined using addressable appli-
cation resources. Therefore an implementation of a service
can determine the active state configuration by querying the
application state. There is no need to keep any additional
protocol state.

Determining what is the active state configuration of the
interface state machine every time that a service implementa-
tion has to fulfill a request may be a slow task in the case
of complex interfaces with many states. However, in practice
it is not necessary to explore all states of the transitions that can be
triggered based on the current request. We show in the next
section how we can do that by computing the precondition
(and postcondition) of each method request.

4.2 Mapping Protocol State Machines to Method
pre- and Postconditions

In this section we show how to extract the contract in-
formation from a UML protocol state machine with state

![Diagram of HRB RESTful Web Service](image-url)
The contract contains the preconditions for each method that triggers a transition in the behavioral model.

The precondition of a method states under what conditions a method can be triggered. We say that the precondition of a method $m$ is satisfied when the state invariants of all the source states of transition $t$ are true along with its guard condition.

In a similar manner, if a method $m$ triggers a transition $t$ in a behavioral model, then its post-condition is satisfied when the state invariants of all the target states of transition $t$ are true along with the post-condition annotated on the transition $t$.

We use the formalization of the structure and semantics of the UML protocol state machine presented in [16]. This formalization supports formal definitions for generating pre- and post-conditions for class methods. We extend this work by generating contracts for HTTP methods from our behavioral models for RESTful web services.

In order to shorten the description of the contract we use path variables to represent the address of a resource. First, the precondition for a method that triggers a transition in the behavioral model is presented. The precondition of a method $m$ is given by taking into account all the transitions that are triggered by $m$. If it is a simple transition, then the state invariant of its source state is conjuncted with the guard of the transition. In case the transition is a trigger to more than one transition, with true guards, and all the transitions have different source states, then the precondition is given by taking a disjunction of state invariants of all the different source states. This implies that the method can trigger a transition whenever it is in one of its source states.

A transition can occur from one state to another if the method that triggers this transition is invoked and its precondition is true. For the transition to be successful, the postcondition of the transition should also be true after the method is invoked. This is specified by the implication operator that relates a precondition of a transition with its postcondition.

A postcondition for a method is extracted from the protocol state machine by manipulating the state invariants of the target states of transitions and the post-conditions on transitions. The post-condition of a fork transition, with true postcondition, specifies that the state invariants of all its target states are true and for a self-transition, its post-condition ensures that the same state invariants are true that were true before invoking the HTTP method.

For the details and formal definitions of generating preconditions and postconditions for different elements in a UML protocol state machine of a class readers are referred to [16].

The postcondition of a transition will be evaluated only if the precondition for that transition is true. We define as
pre_OK(r) the function that gives boolean value of OK(r) on resource r before invoking the trigger method. Similarly, pre.pc.waiting, pre.pc.confirm and pre_NOT_FOUND(r) give the representation of pconfirmation and boolean value of NOT_FOUND(r) before invoking the trigger method, respectively.

The excerpt below from the list of high-level contracts generated from Figure 2 shows the contracts generated for the HTTP method POST on cancel resource.

```
PATH
b: bookings/{bid}/
r: bookings/{bid}/rooms/{rid}
pre: bookings/{bid}/payment/pconfirmation
c: bookings/{bid}/cancel
POST { bookings/{bid}/cancel

precondition
{OK(b) && OK(r) && NOT_FOUND(c) } &&
{ NOT_FOUND(pc) || OK(pc) && pc.waiting == false }

postcondition
{ pre_OK(b) && pre_OK(r) && pre_NOT_FOUND(c) } &&
{ [preNOT_FOUND(pc) == OK(b) && OK(c)] } &&
{ (pre_OK(pc) && pre.pc.waiting == false) ==>
  OK(b) && OK(c) }
```

The conceptual model as shown in Figure 1 and behavioral model as show in Figure 2 lead to RESTful interfaces. This is explained further in the next section.

5. GENERATION OF BEHAVIORAL WADL SERVICE DESCRIPTIONS

Besides the UML representation of the behavior interfaces, we propose to extend the Web Application Description Language (WADL) to include information about the behavior of the methods in a service. Our objective is that this information is generated automatically from the conceptual and behavioral models described before.

WADL defines the operations that can be invoked on an interface and describes the input and output parameters for each operation. It defines the resources that an application contains and methods that can be called on them. Each method has two attributes name and id, where name is the name of the HTTP method and id is the ID of the method that is associated with the HTTP method.

Representing the information in the conceptual model as part of a WADL service description is a rather straightforward task. However, the behavioral model does not map directly to a WADL description since the behavioral model allows different transitions to be trigger by the same method. In our example, a cancel request can be invoked when the application is in different states. That is the information about when a method can be invoked (precondition) and what is its result (postcondition) needs to be computed from the different states in the behavioral model.

5.1 Inserting Pre- and Postconditions into WADL Service Descriptions

The high-level contracts are refined with details on relative navigation paths, invoked HTTP methods and the expected response codes. These refined contracts are asserted into WADL interface. The function pre_OK(r) is mapped to a pre_GET function and its response code is compared to 200. The pre_GET(r) function gives the stored results of invoking a GET method on resource r before invoking the method. In similar manner, a pre_NOT_FOUND is mapped to a pre_GET function and its response code is compared to 404.

In order to support the behavioral information in the interface descriptions, we have extended the XML schema of WADL with two elements precondition and postcondition, with an attribute id for each of these elements. These tags, i.e., <precondition> and <postcondition> are asserted above and under the method tag, respectively. An excerpt of a behavioral RESTful interface is shown below for method POST on cancel resource.

```
<resources base = "http://www.example.com/bookings">
  <resource path = "{bid}"/>
    <resource path = "cancel">
      <precondition id = "pre_post_cancel" >
        (GET(/bookings/{bid}) == status(200) &&
        GET(/bookings/{bid}/room/{rid}) == status(200) &&
        GET(/bookings/{bid}/cancel) == status(404) ) &&
        (GET(/bookings/{bid}/payment/pconfirmation) == status(200) &&
        GET(/bookings/{bid}/payment/pconfirmation#waiting == false) ==>
          GET(/bookings/{bid}/cancel) == status(200) &&
        GET(/bookings/{bid}/payment/pconfirmation#waiting == false) ==>
        GET(/bookings/{bid}/cancel) == status(200) &&
      </precondition>
      <postcondition id = "post_post_cancel" >
        (pre_get(/bookings/{bid}) == status(200) &&
        GET(/bookings/{bid}/room/{rid}) == status(200) &&
        GET(/bookings/{bid}/cancel) == status(404) ) &&
        ( GET(/bookings/{bid}/payment/pconfirmation) == status(200) &&
        GET(/bookings/{bid}/cancel) == status(200) &&
        GET(/bookings/{bid}/payment/pconfirmation#waiting == false) ==>
        GET(/bookings/{bid}/cancel) == status(200) &&
      </postcondition>
    </resource>
  </resource>
</resources>
```

6. RESTFUL BEHAVIORAL INTERFACES

As discussed earlier, a REST interface should exhibit these four attributes: addressability, connectivitiy, statelessness and uniform interface. We claim that we can create RESTful interface following the approach described in this paper. Our conceptual resource model and behavioral model lead to web services that exhibit these attributes and make our interfaces RESTful by construction.

We constrain our conceptual model to be a connected graph such that no resource is isolated. The associations between classes in the conceptual model provide connectivitiy between the resources. Each resource can be addressed independently using the navigation directions of associations and their role names. The role names show the relative navigation path between the resources. For example, the navigation path to a room that belongs to a particular booking ID(bid) is given as /bookings/{bid}/rooms/{rid}. Thus, each resource has a URI address providing addressability feature to our conceptual resource model.
Unlike software class diagrams showing static structure of RPC web services, our conceptual model does not contain any method information. We restrict the behavioral model so that transitions can only be triggered using the standard HTTP methods, providing the uniform interface feature.

In addition, we have created stateful service using stateless service interface thanks to the fact that the application states are defined using state invariants that are defined in terms of exposed resources. This information is presented in the behavioral model of the REST interface, providing the statelessness feature in our behavioral model.

7. APPLICATIONS OF BEHAVIORAL INTERFACES

Service descriptions are often used to automatically generate code stubs to invoke the service from a particular programming language. Another interesting application is the creation of a service register to publish and discover web services. Still, a service description enriched with behavioral contracts has many other applications that we describe below.

A web service developer can use the behavioral REST interface as a specification to implement the web service. The UML protocol state machines do not contain executable actions, unlike behavioral state machines, and hence are not executable. On the other hand, they provide rich behavioral information of an interface. Developers implementing a web service have to manually implement the interface specified in protocol state machine. This requires efforts to ensure that implementation conforms to its behavioral specification. Our approach can be extended to generate implementations stubs from the presented models in an automated manner. In addition, behavioral specifications can be automatically generated from models and asserted as contracts into programmatic interface of the web service. This work is under development using Django framework.

A service implementation can use the asserted contracts to validate a request from the client. The preconditions of a method provide a check on the incoming request. Thus, ensuring whether the conditions to invoke a service method are met before invoking the method can be an efficient activity in terms of cost and bandwidth. Similarly, a client can benefit from the asserted postconditions to validate a response from the server, constraining the provider of the service to ensure the functionality that is expected from it.

In order to validate a service, test cases can be generated from the behaviorally enriched interfaces. These test cases can validate the implementation of a web service and these contracts can also be exploited for the generation of test oracles. Test oracles are used to determine whether a test has passed or failed. In the context of test case generation and test oracle generation, we can take advantage of several efforts done previously to validate the behavior of classes and web services using contracts [8] [7].

We can also use these behavioral web service interfaces to provide a monitoring mechanism. The behavioral specifications can be added as a proxy interface to already developed and deployed web services to monitor their functioning. This facilitates location of the fault in and application by observing the conditions that are not being met and by which methods. It can also check for any failure caused by a network fault, late delivery or if a service developer violates a certain pre or post condition of a method by mistake during implementation.

8. RELATED WORK

Several authors have used UML class diagram and other UML behavioral diagrams to model RPC styled web services and their compositions [17].

In terms of modeling behavioral specifications of web services, Bertolino and Polini have used UML protocol state machines for specifying the intended protocol of services [2]. Protocol state machines are transformed to Symbolic Transition System that is extended with Labeled Transition System(LTS) semantics.

Modeling of RESTful web services has been addressed in the work of Systa and her research group [11, 12, 19]. In [11, 12], an approach that migrates legacy APIs to RESTful web services is presented. In [12], Laitkorpi et al. present the process that transforms service functionality over different phases to a RESTful service. This approach is further explored in [19] by Siikala et al. They provide an iterative and incremental process for the development of model transformations by focusing on the step of transforming information model to resource model.

The role of contracts in the domain of web services has been investigated previously, e.g., [6] [4], etc. In [6], Castagna et al. present theory of contracts that formalizes the compatibility of a client to a service and introduces a subcontract relation for behavioral typing of web services promoting service reuse or redefinition. In [3] and [4], a theory of contract is presented that addresses the problem of composition of multiple services. The correctness for service compositions is modeled using process calculi and the notions of strong service compliance and strong subcontract pre-order are investigated.

For REST architectural style, Zou et al. [21] propose an Accountable State Transfer(AST) architecture to bridge the accountability gap in REST. The approach uses service contract as foundation for enabling accountability. The paper outlines the architectural decisions and principles for enabling accountability, followed by AST architecture with accountable state transfer protocol. The service contracts are implemented as an ontology and the service executions are stored in a knowledge base to allow reasoning between different concepts of a contract.

In the context of modeling behavioral specifications and using contracts with UML, Lohmann et al. [14, 10], use visual contracts to specify the dynamic behavior and class diagrams to specify the static aspect of a web service. Graph transformations are annotated on to the class diagram with object diagrams specifying pre- and post-conditions of the operations.

We consider that the main strength of our approach is that our models lead to RESTful web services. In addition, we can generate implementation stubs and behaviorally enriched interfaces from models that have many applications. The use of UML protocol state machine for modeling provides behaviorally enriched interfaces that show the sequence of operations that can be invoked on the service, the conditions under which they can be invoked and the expected results. Both the conceptual and behavioral models serve as specification document. Also, contracts can be directly gen-
erated from these models and asserted into syntactic service descriptions as shown in Section 4 and 5.

9. CONCLUSION AND FUTURE WORK

In this article, we present a novel approach to model the behavioral interface of a RESTful web service. A conceptual resource model shows different resources and how can they be addressed. The behavioral model specifies how the service should be used by showing the order of method invocations and the conditions on these methods. We have used UML class and protocol state machine diagrams to model the conceptual and behavioral aspects of the web service. The models can be used to generate a contract in the form of preconditions and postconditions for methods of an interface. These contracts can be included in a WADL interface specification. The resulting service is RESTful by construction.

The behavioral RESTful interfaces have many applications. They can serve as documentation for existing services or as a blueprint for developing new ones. The models can be used to generate implementation stubs in commonly used web frameworks like Django and Ruby on Rails. They can also be used to monitor the interaction between a service and its clients and warn if any of the parties breaches the interface contract. The generation of tests from interface contracts is also a promising application of formalized behavioral interfaces.

In the context of web services, our work is novel as we provide a modeling approach for a RESTful web service that captures both the static and behavioral features of an interface. The models exhibit all attributes of REST web services and are easy to understand and communicate.

We are currently working on extending our approach to support modeling interfaces for composite RESTful services and provide a complete modeling approach that also caters the process of RESTful service composition.

10. REFERENCES