From Algebraic Specification to Ontological Description of Service Semantics

Dongmei Liu
School of Computer Science and Technology
Nanjing University of Science and Technology
Nanjing, 210094, P.R. China
dmliukz@njust.edu.cn

Hong Zhu and Ian Bayley
Dept of Computing and Communication Technologies
Oxford Brookes University
Oxford OX33 1HX, UK
hzhu@brookes.ac.uk, ibayley@brookes.ac.uk

***Abstract*—The accurate description of service semantics plays a crucial role in service discovery, composition and interaction. Most work in this area has been focussed on ontological descriptions, which are searchable and machine-understandable, but do not define service functionality in a verifiable and testable way. Formal specification techniques, having evolved over the past 30 years, can define semantics verifiably and testablly, but they have not yet been applied to service computing because formal specifications are not searchable. There is a huge gap between these two methods of semantics description.**

In this paper, we bridge the gap. Our technique is to specify services formally in an algebraic specification language and then to extract ontological description as profiles in the language OWL-S, with the associated searchability benefits. We present a prototype tool for performing this transformation and report a case study to demonstrate the feasibility of our approach. The algebraic specification language we use is SOFIA (Service Oriented Formalism in Algebras).

***Keywords*-Web services, Formal semantics, Algebraic specification, Ontology, OWL-S.**

I. Introduction

The advent of Web Services as autonomous, platform-independent computational entities has greatly facilitated the uptake and use of the paradigm of service-oriented computing. Various initiatives have been advanced to define the semantics of services. These are aimed at enabling the automation of service discovery, composition and interoperation. Generally speaking, the semantics of software systems and their components can be described in two different ways: using ontologies and using formal specifications. The former is easy for software developers to understand and for computers to process. The latter, however, gives semantics that are both testable and verifiable. Unfortunately, there is wide gap between these two approaches.

This paper aims at bridging the gap with an automated software tool that converts the formal specification to the ontology, thereby conferring the machine-readability and human-understandability benefits of ontologies onto formal specifications. The main contributions of this paper are:

1) a set of mapping rules that translate algebraic specifications into domain ontologies,
2) the rules that, given such a domain ontology, extract ontological descriptions of services from the same algebraic specifications,
3) a tool called *TrS2O* that implements both rules for the SOFIA algebraic specification language, and
4) a case study in which the tool is applied to a real world RESTful Web Service, the API of GoGrid [1].

The GoGrid API is a RESTful Web Service that enables resource management of Infrastructure-as-a-Service (IaaS) on the cloud. SOFIA has been devised as an improvement on our previous algebraic specification language CASSOC-WS [2], with significant changes to the syntax and semantics. For the sake of space, SOFIA will be documented in a separate paper. A short reference manual is available at [3]. The transformation rules, however, will be presented in a general language-neutral form though the tool TrS2O has been implemented for SOFIA.

The remainder of the paper is organised as follows. Section II defines the general mathematical structure of algebraic/co-algebraic specifications. Section III presents the mapping rules that translate algebraic specifications into ontologies and the rules that extracts ontological descriptions of the service semantics. Section IV describes the prototype tool TrS2O that implements both sets of rules for the SOFIA language and represents the resulting ontology and service semantics in OWL and OWL-S profiles. Section V reports the case study of GoGrid API. Section VI concludes the paper with a comparison of related work and a discussion of future work.

II. Mathematical Structure of Algebraic Specifications

In this section, we integrate the theories of algebraic and co-algebraic specifications to define the mathematical framework in which specification languages for service-oriented systems, such as SOFIA, can be formally defined.

A. Algebraic Structures

We regard a service-oriented system as consisting of a collection of units. As in [4], we assume that the specification of a software system is well-structured in the sense

that each software unit has a corresponding specification unit with a unique *sort* name, and any real-world concepts and entities that are used are also specified by corresponding specification units.

We recognise two different ways in which one unit can be constructed from another, extension and usage, as follows:

- a unit can be extended with additional elements, in a manner similar to the inheritance relation of object-orientation. The notation $s' \rhd s$ means that s' extends s.
- a unit can use another unit eg as a component, or as a parameter or result from an operation etc, like the association relation of object-orientation. The notation $s' \prec s$ means that s uses s', and $s' \preceq s$ means that $s' \prec s$ or $s' = s$.

A well-structured specification of a collection of software units should also preserve both of these relationships between units. Formally, we define the notion of *system signature* to represent the overall structures of software systems as follows.

Definition 1: (System signature) A *system signature* is an ordered pair $(\mathbf{S}, \mathbf{\Sigma})$, where $\mathbf{S} = \langle S, \rhd, \prec \rangle$ is a set S of sorts with two binary relations on S denoted by $\rhd$ and $\prec$, and $\mathbf{\Sigma} = \{\Sigma^s | s \in S\}$ is a collection of unit signatures, with Σ^s denoting the unit signature for sort s. □

The notion of unit signature will be defined below after the necessary preliminaries.

Every kind of software entity, whether it be an abstract data type, a class, a component or, as here, a service, must define a set of typed operators. The syntactic aspect of an operator is determined by its domain, its co-domain and its identifier and is specified in the following form.

$$op : s_1, s_2, \cdots, s_n \rightarrow s'_1, s'_2, \cdots, s'_k$$

where *op* is the identifier of the operator, $(s_1, s_2, \cdots, s_n)$, $n \geq 0$, are the domain sorts, and $(s'_1, s'_2, \cdots, s'_k)$, $k > 0$, are the co-domain sorts.

We allow an operator to have more than one domain sort and more than one co-domain sort at the same time. This is the main difference between our theory and that used for algebraic specifications, which require a single sort co-domain, and that used for coalgebraic specifications, which require a single sort domain. These restrictions are too tight to specify services so they are relaxed in our theory. This allows us, for example, to give a `BookTicket` operator for an online ticket booking service a signature like this:

```
BookTicket: DATE,NAT,BOOKING -> MESSAGE,BOOKING.
```

Here `DATE` is the date of the performance, `NAT` is the number of tickets wanted, `MESSAGE` is the response to the requester. `BOOKING` represents the state of the online booking services. It occurs in both the domain and the co-domain so that the original state can be taken as input and the modified state can be produced as output.

We now define the notion of *unit signature* to represent the structure of software units, as follows.

Definition 2: (Unit Signature)

Given a system signature $(\mathbf{S}, \mathbf{\Sigma})$, the *unit signature* for a sort $s \in \mathbf{S}$, denoted by Σ^s, consists of a finite family of disjoint sets $\Sigma^s_{w,w'}$ indexed by pairs of units (w, w') with w and $w' \in W_s = \{x \in S | x \prec s \vee x = s\}^*$. Each element φ in set $\Sigma^s_{w,w'}$ is an *operator symbol* of type $w \rightarrow w'$, where w is the *domain type* and w' the *co-domain type* of the operator. □

Such operators can be classified as *constants*, *variables*, or *general operations* as follows.

1) φ is a *constant*, if $w = \emptyset$, $w' = (s)$,
2) φ is a *variable*, if $w = (s)$, $w' = (s')$, and $s' \prec s$,
3) Otherwise, φ is a *general operation*.

In the sequel, we will write Σ^s_C, Σ^s_V and Σ^s_G for the subsets of Σ^s that contain the constants, the variables and the general operations, respectively.

The semantics of the operators are defined by axioms that describe the properties that these functions must satisfy. An axiom consists of a number of universally quantified variables and a list of conditional equations. Terms of sort s are called *s-terms*.

Let $(\mathbf{S}, \mathbf{\Sigma})$ be a system signature, $\{V_s | s \in S\}$ be a collection of disjoint sets of variables, where elements of V_s are called variables of sort s, and $s \in S$ be any given sort. Then, we have:

Definition 3: (Terms)

The set of *s-terms* is inductively defined as follows. Let $s_1, \cdots, s_n \preceq s$. Then

1) An s'-term τ of type w is a s-term of type w, if $s' \prec s$.
2) A variable $v \in V_{s'}$ is a s-term of type s'.
3) $\langle \tau_1, \cdots, \tau_n \rangle$ is a s-term of type $(s_1, \cdots, s_n)$, if $\tau_1, \cdots, \tau_n$ are s-terms of types $s_1, \cdots, s_n$, respectively.
4) $\varphi(\tau)$ is a s-term of type w', if τ is an s-term of type w and $\varphi^{s'} : w \rightarrow w'$ is an operator in the unit signature $\Sigma^{s'}$ for sort $s' \preceq s$.
5) $\tau \# K$ is a s-term of type s_K, if τ is an s-term of type $(s_1, \cdots, s_n)$, where $K \in \{1, \cdots, n\}$. □

Next, equations and axioms are defined as follows.

Definition 4: (Equation)

Let τ and τ' be s-terms of type w, and let $c_1, \cdots, c_n$ and $d_1, \cdots, d_n$ be s-terms such that for all $i = 1, \cdots, n$, c_i and d_i are of type $s_i \preceq s$. A *conditional equation* e of signature Σ^s has the form

$$\tau = \tau', \text{ if } c_1 = d_1, \cdots, c_n = d_n.$$

The condition is optional, and when omitted, the above is called *unconditional equation*, or simply an *equation*. □

Definition 5: (Axiom)

An axiom ax is an ordered pair (V_{ax}, E_{ax}), where E_{ax} is a list of conditional equations, V_{ax} is a set of variables

that occur in the equations. These are universally quantified at the outermost. □

A specification of a software unit consists of a unit signature and a set of axioms, whereas a specification of a software system consists of a set of specification units, which form a system signature and a set of axioms. Formally, we have the following definition:

Definition 6: (Specification)

A *specification* is a triple $(\mathbf{S}, \mathbf{\Sigma}, \mathbf{Ax})$, where

1) $\mathbf{S} = \langle S, \triangleright, \prec \rangle$, S is a finite set of sorts, $\prec$ and $\triangleright$ are binary relations on S that represent the uses and inheritance relations, respectively;
2) $\mathbf{\Sigma} = \{\Sigma^s | s \in S\}$ is a set of unit signatures indexed by S,
3) $\mathbf{Ax} = \{Ax^s | s \in S\}$ is finite set of axioms indexed by S,
4) for all s and $s' \in S$, $s' \triangleright s$ implies that $\Sigma^s \subseteq \Sigma^{s'}$ and $Ax^s \subseteq Ax^{s'}$.

For each $s \in S$, (Σ^s, Ax^s) is called the *specification unit* for sort s. □

Note that, by Definition 2, a specification consists of a system signature $(\mathbf{S}, \mathbf{\Sigma})$, and a collection $\mathbf{Ax}$ of axiom sets.

B. Semantics of Algebraic Structures

We now define the semantics of algebraic specifications by defining what is a correct implementation of a specification. In general, an implementation of a specification is a mathematical structure that realises the operators specified in the signature with operations that satisfy the axioms.

Definition 7: (Algebra)

Given a system signature $(\mathbf{S}, \mathbf{\Sigma})$, a $(\mathbf{S}, \mathbf{\Sigma})$-*algebra* $\mathcal{A}$ is a mathematical structure $(\mathbf{A}, \mathbf{F})$ that consists of a collection $\mathbf{A} = \{A_s | s \in S\}$ of sets indexed by S, and a collection $\mathbf{F}$ of functions indexed by the set $\bigcup_{s \in S} \Sigma^s$ such that for each operator $\varphi^s : w \to w'$, the function $f_\varphi \in \mathbf{F}$ has domain A_w and co-domain $A_{w'}$, where $w = (s_1, s_2, \cdots, s_n)$, $A_w = A_{s_1} \times \cdots \times A_{s_n}$, $w' = (s'_1, s'_2, \cdots, s'_n)$, and $A_{w'} = A_{s'_1} \times \cdots \times A_{s'_n}$. □

The evaluation of a term in an algebra depends on the values assigned to the variables that occur in the term. Such an assignment α of variables V_s, $s \in S$, in an algebra $\mathcal{A}$ is a function from V_s to A_s.

Definition 8: (Evaluation of terms in an algebra)

Given an assignment α. The evaluation of a term τ in an $(\mathbf{S}, \mathbf{\Sigma})$-algebra $\mathcal{A} = (\mathbf{A}, \mathbf{F})$, written $Eva_\alpha(\tau)$, is defined as follows.

1) $Eva_\alpha(v) = \alpha(v)$;
2) $Eva_\alpha(\langle \tau_1, \cdots, \tau_n \rangle) = \langle Eva_\alpha(\tau_1), \cdots, Eva_\alpha(\tau_n) \rangle$;
3) $Eva_\alpha(\varphi(\tau)) = f_{A,\varphi}(Eva_\alpha(\tau))$.
4) $Eva_\alpha(\tau \# K) = V_K$, where $Eva_\alpha(\tau) = \langle V_1, \cdots, V_n \rangle$, $1 \leq K \leq n$. □

Let e be an equation in the following form.

$$\tau = \tau', \ \textbf{if } c_1 = d_1, \cdots, c_n = d_n.$$

Definition 9: (Satisfaction)

An $(\mathbf{S}, \mathbf{\Sigma})$-algebra $\mathcal{A} = (\mathbf{A}, \mathbf{F})$ *satisfies* e, written $\mathcal{A} \models e$, if for all assignments α, we have that $Eva_\alpha(\tau) = Eva_\alpha(\tau')$ whenever $Eva_\alpha(c_i) = Eva_\alpha(d_i)$ is true for all $i = 1, 2, \cdots, n$.

Let $\mathcal{E} = (\mathbf{S}, \mathbf{\Sigma}, \mathbf{Ax})$ be a specification. An $(\mathbf{S}, \mathbf{\Sigma})$-algebra $\mathcal{A} = (\mathbf{A}, \mathbf{F})$ satisfies specification $\mathcal{E}$, written $\mathcal{A} \models \mathcal{E}$, if for all equations e in $\mathbf{Ax}$, we have that $\mathcal{A} \models e$. □

III. Mapping Specification To Ontology

An ontology is a description of the concepts in a domain through the relations between the concepts, and the individuals as the instances of the concepts and relations. In ontology modelling languages, such as OWL, the concepts are often modelled by *classes*, and the relations are modelled by *properties*, which are also used to describe the features and attributes of the concepts. The individuals are modelled by objects, i.e. instances of a concept. An ontology that consists of a set of concepts and relationships among them together with a set of individual instances is a representation of domain knowledge [6].

In this section, we present a set of mapping rules to derive ontological descriptions of services from algebraic specifications.

A. Extraction of Domain Ontology

Given a specification $(\mathbf{S}, \mathbf{\Sigma}, \mathbf{Ax})$ of service Sv, the following set of rules will translate the specification into an ontology. These rules extract classes, properties and individuals from algebraic specifications.

Rule 1: For each sort $s \in S$ of the specification, generate a formula $Class(s)$. □

The predicate $Class(x)$ means that x is a class or concept.

Rule 2: For an extension relation $s' \triangleright s$ in the system signature $(\mathbf{S}, \mathbf{\Sigma})$ of the specification, generate a formula $subClassof(s', s)$. □

The predicate $subClassof(x, y)$ means that class x is a subclass of y, or equally, x is a sub-concept of y.

Rule 3: For a uses relation $s' \prec s$ in the system signature $(\mathbf{S}, \mathbf{\Sigma})$ of the specification, generate a formula $uses(s, s')$. □

The predicate $uses(x, y)$ means that concept x is defined using the concept y. The uses relationship between classes or concepts is somehow redundant because further details about how x uses y will be presented in other precidates deduced from the operations on the sorts.

Rule 4: For each operator $\varphi : s \to s' \in \Sigma^s_V$,

1) A formula $Property(\varphi)$ is generated,
2) A formula $\varphi(s, s')$ is generated. □

Informally, the predicate $Property(\varphi)$ means that φ is a property. The predicate $\varphi(x, y)$ means that φ is a property of concept x (i.e. an attribute or an element of x), and its value is of type y.

Rule 5: For each general operation $\varphi : w \to w' \in \Sigma^s_G$,

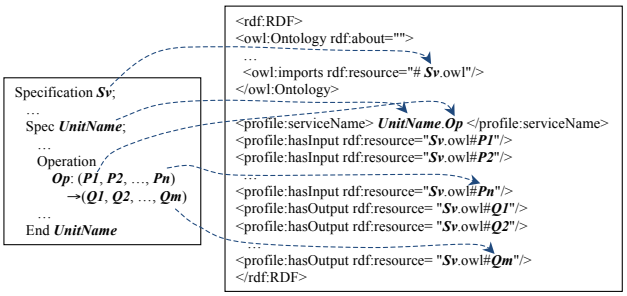

Figure 1. Rules to Translate Specification Unit into Service Profile

1) Generate a formula $Class(\varphi)$.
2) For each $s \in w$, generate a formula $Domain(\varphi, s)$.
3) For each $s \in w'$, generate a formula $Codomain(\varphi, s')$. □

Note that the formula $Class(\varphi)$ means that φ is a concept. Here, we regard an operation as a relation (i.e. a relational concept) that links the concepts of the domain to the concepts of the co-domain. The predicate $Domain(\varphi, x)$ means that x is a domain of the relation φ. The predicate $Codomain(\varphi, x)$ means that x is a co-domain (or range or output) of the relation φ.

Rule 6: For each constant $\varphi \in \Sigma^s_C$,

1) generate a formula $Individual(\varphi)$, and
2) generate a formula $s(\varphi)$. □

The predicate $Individual(y)$ means that y is an individual and $x(y)$ means that y is an instance of class x.

B. Generation of Service Profile

Having generated the ontology from a specification, the services can be described in OWL-S profiles based on the ontology. Such a profile can also be generated from the specification unit that defines the service's functionality. Figure 1 shows the mapping rules between specifications and service profiles.

For example, the following is the specification unit that defines the operations on Servers in the GoGrid system, where the axioms are omitted since they are not used in the translation.

```
Spec GServer;
  Uses
   ServerListRequest, ServerListResponse,
   ServerGetRequest, ServerGetResponse,
   ServerAddRequest, ServerAddResponse,
   ServerEditRequest, ServerEditResponse,
   ServerDeleteRequest, ServerDeleteResponse,
   ServerPowerRequest, ServerPowerResponse;
  Var
   clockTime: Integer;
  Operation
   List(GServer, ServerListRequest) :
     GServer, ServerListResponse;
   Get(GServer, ServerGetRequest) :
     GServer, ServerGetResponse;
   Add(GServer, ServerAddRequest) :
     GServer, ServerAddResponse;
   Edit(GServer, ServerEditRequest) :
     GServer, ServerEditResponse;
   Delete(GServer, ServerDeleteRequest) :
     GServer, ServerDeleteResponse;
   Power(GServer, ServerPowerRequest) :
     GServer, ServerPowerResponse;
  Axiom
   ...
End
```

It can be translated into the functional part of the service profile. A fragment of the profile for the List operation is given below.

```
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource=
     "http://www.daml.org/services/owl-s/1.0/
       Profile.owl"/>
  <owl:imports rdf:resource=
    "#GServerOntology.owl"/>
</owl:Ontology>
<profile:serviceName> GServer.List
</profile:serviceName>
<profile:hasInput rdf:resource=
  "GServerOntology.owl#GServer"/>
<profile:hasInput rdf:resource=
  "GServerOntology.owl#ServerListRequest"/>
<profile:hasOutput rdf:resource=
  "GServerOntology.owl#GServer"/>
<profile:hasOutput rdf:resource=
  "GServerOntology.owl#ServerListResponse"/>
</rdf:RDF>
```

IV. Tool TrS2O

A prototype tool called TrS2O (**Tr**anslator from **S**pecification **t**o **O**ntology) has been designed and implemented in Java for translating formal specifications in SOFIA to ontological descriptions of services in OWL. Figure 2 shows the overall structure of TrS2O Tool.

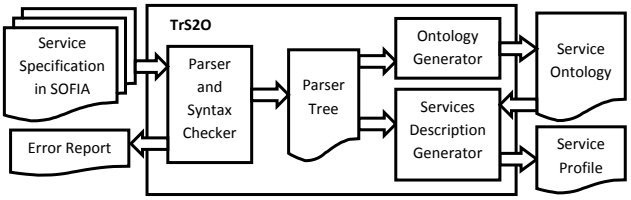

Figure 2. Overall Structure of TrS2O Tool

The tool TrS2O contains three main components.

- *Specification Parser and Syntax Checker*: It parses algebraic specifications written in SOFIA and generates a parse tree. It checks the well-formedness of the specifications and type correctness of equations in the axioms.
- *Ontology Generator*: It takes the parse tree of the algebraic specification as input, and generates an ontology

represented in OWL language according to the rules defines in section III.

- *Services Description Generator*: It takes the parser tree of the algebraic specification and the ontology as input, and generates the descriptions of services in OWL-S profiles.

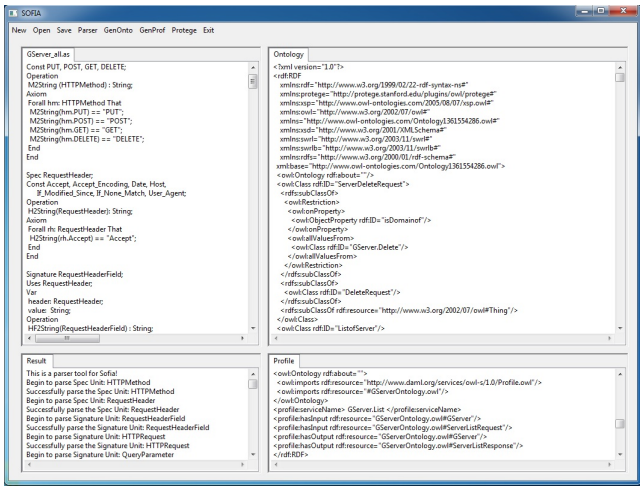

Figure 3. Interface of TrS2O Tool

Figure 3 shows the interface of the tool, where on the left hand side of the window, the upper part displays the specification in SOFIA; the lower part is the parsing report. On the right hand side, the upper part displays the ontology generated by TrS2O, and the lower part is the profile of services generated from the specification. It is worth noting that the ontology generated by TrS2O can be processed by any OWL tool, such as Protege[1] for visualization, reasoning, and search for domain knowledge based on ontology.

V. Case Study

In this section, we report a case study using a real industrial RESTful web services GoGrid.

A. GoGrid API

GoGrid is an infrastructure-as-a-service (IaaS) provider [1]. It provides an easy-to-use API for developers, systems administrators and end-users to access its functions. Its services can be accessed through a RESTful web service interface in a number of different programming and scripting languages. RESTful web services, unlike SOAP/WSDL, are based on the HTTP protocol, so each GoGrid API call is an individual HTTP query.

GoGrid API has 5 types of common requests: *List*, *Get*, *Add*, *Delete*, and *Edit*. These types of requests can be applied to 8 different types of objects: *Job*, *Load balancer*, *Server*, *Image*, *IP*, *Passwords*, *Billing* and *Utility*. Some of the requests are not applicable to all types of objects, while some objects have special operators. Table I gives the applicable operators for each type of objects.

Table I
Applicable Operators on Objects

Object	List	Get	Add	Delete	Edit	Other Ops
Job	Yes	Yes				
Load Balancer	Yes	Yes	Yes	Yes	Yes	
Server	Yes	Yes	Yes	Yes	Yes	Power
Server image	Yes	Yes		Yes	Yes	Save, Restore
IP	Yes					
Password	Yes	Yes				
Billing		Yes				
Untility	Yes					

B. GoGrid Specification in SOFIA

In the case study, we first formally specify GoGrid API in SOFIA. For each type of objects of GoGrid API, the formal specification in SOFIA consists of three types of specification units:

- units that specify the valid requests, including their structures and the constraints on the combinations of their components;
- units that specify the responses, also including their structures and the constraints on the valid combinations of their components;
- units that specify the objects of certain types, including their structures in terms of the signatures and the semantics of the operations expressed in axioms that characterise the relationships between the requests and the responses.

There are also some specification units that define the features and concepts that are common to many types of objects, requests, or responses. For example, there are four query parameters that are common to all GoGrid API calls. For each type of request, there are also some properties that are common to all types of objects.

The specification of GoGrid is based on a framework for specifying RESTful web services. The framework defines the common structure and features of all RESTful web services, such as the structure of HTTP requests and responses as a set of specification units in SOFIA. A concrete RESTful web services can be specified in a number of specification units that extend the framework units. For example, the following is the specification unit that defines *HTTPRequest*.

```
Spec HTTPRequest;
 Uses HTTPMethod, RequestHeaderField;
 Var
     method: HTTPMethod;
     server: String;
     path:   String;
     headerFields[1..*]: RequestHeaderField;
     body:   String;
End
```

[1]http://protege.stanford.edu/

Note that, in SOFIA, when a type of objects is structural, i.e., it consists of a number of attributes, each attribute is defined as a variable. In the terminology of algebraic specification, a "variable" is an observer, which is an operation from the sort being defined to another sort. It is similar to the getters in object-oriented programs for getting the value of attributes.

Table II gives the numbers of specification units in GoGrid Specification in SOFIA.

Table II
NUMBER OF UNITS IN GOGRID SPECIFICATION

Type of unit	No.
Framework of RESTful web service	10
Common features	37
Definition of Server operations	13
Definition of Server image operations	13
Definition of Load Balancer operations	11
Definition of Job operations	5
Definition of operations on other objects	14
Total	**103**

C. GoGrid Ontology

Using TrS2O tool, we have extracted ontology from GoGrid specification. Figure 4 shows the ontology generated from the framework part of the specification.

Take HTTPRequest sort for example, the fragment of ontology profile is given below. It includes one class with 2 properties for uses relation and 5 properties for variables defined as ObjectProperty.

```
<owl:Class rdf:ID="HTTPRequest">
 <rdfs:subClassOf rdf:resource=
   "http://www.w3.org/2002/07/owl#Thing"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#uses"/>
   <owl:allValuesFrom>
    <owl:Class rdf:ID="HTTPMethod"/>
   </owl:allValuesFrom>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#uses"/>
   <owl:allValuesFrom>
    <owl:Class rdf:ID="RequestHeaderField"/>
   </owl:allValuesFrom>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:about=
     "#HTTPRequest.method">
 <rdfs:domain rdf:resource="#HTTPRequest"/>
 <rdfs:range rdf:resource="#HTTPMethod"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about=
     "#HTTPRequest.server">
 <rdfs:domain rdf:resource="#HTTPRequest"/>
 <rdfs:range rdf:resource="#String"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about=
     "#HTTPRequest.path">
 <rdfs:domain rdf:resource="#HTTPRequest"/>
 <rdfs:range rdf:resource="#String"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about=
     "#HTTPRequest.headerFields">
 <rdfs:domain rdf:resource="#HTTPRequest"/>
 <rdfs:range rdf:resource=
     "#RequestHeaderField"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about=
     "#HTTPRequest.body">
 <rdfs:domain rdf:resource="#HTTPRequest"/>
 <rdfs:range rdf:resource="#String"/>
</owl:ObjectProperty>
```

VI. CONCLUSION

In this paper, we proposed an approach to bridge the gap between formal specification and ontological description of service semantics by transforming formal specifications into domain onotology and ontological descriptions of services. The former is capable of providing verifiable and testable specifications of service semantics, while the later has the advantage of practical usability and easy to understand for software developers. The prototype tool is built for the specification language SOFIA, and the output is in OWL. However, the set of mapping rules to translate algebraic specifications of service to ontology is general and applicable to all algebraic specification languages and all ontology description languages. A case study with the tool demonstrates the feasibility of the proposed approach.

A. Related Work

OWL-S [7], [8] was the first major ontology definition language for the description of the semantics of Web Services. It provides a set of constructs for describing the properties and capabilities of Web services in a machine-readable format. WSMO [9] (Web Service Modelling Ontology) is a conceptual model for semantic description of Web Services realized in Web Service Modelling Language (WSML) [10]. In addition to the work on the so-called *Big Web Services*, there are also attempts to develop languages and facilities to describe semantics of RESTful web services, such as WADL [11], MicroWSMO/hRESTS [12], and SA-REST [13]. These ontology-based approaches describe the semantics of services using vocabulary defined in an ontology for the meanings of the input and output parameters as well as the functions of the services. Such descriptions have the advantages of easy to understand for human developers and efficient for processing by computers. However, this approach is inadequate to provide verifiable and testable definitions of services' functions, because an ontology can only define a vocabulary through the stereotypes of relationships between the concepts and their instances.

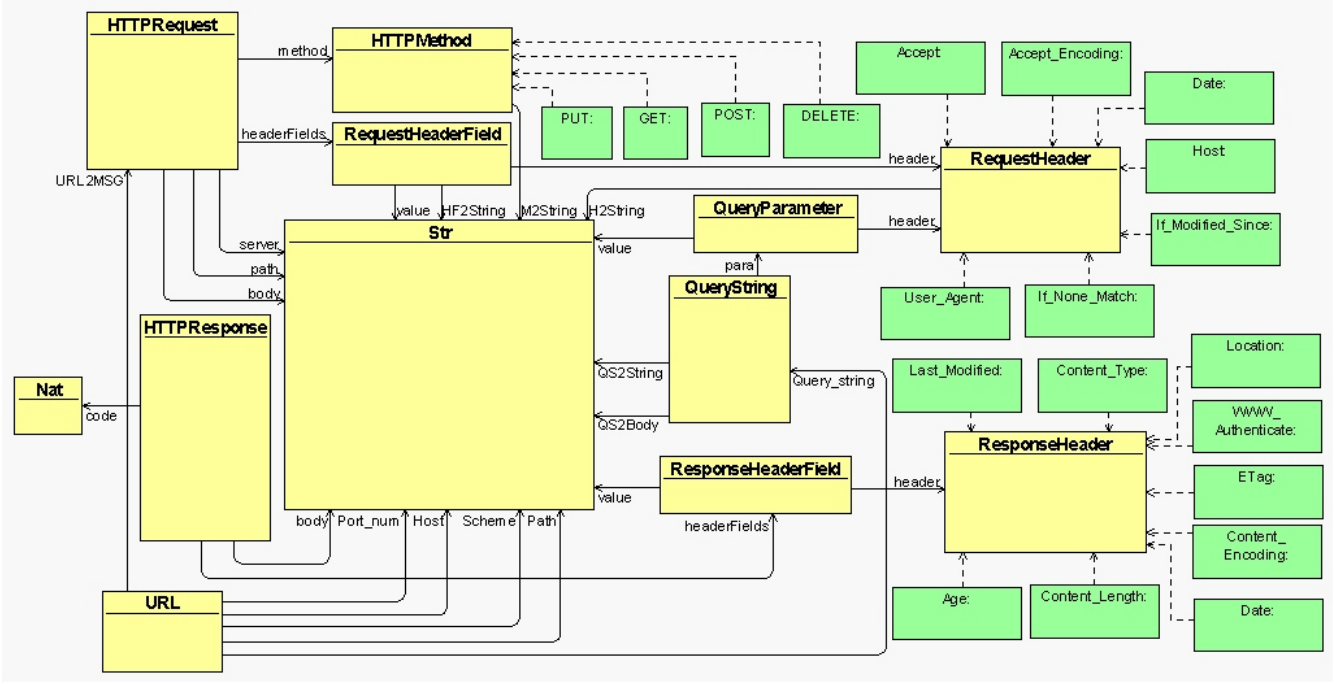

Figure 4. Ontology Generated from the Framework part of GoGrid Specification

Algebraic specification as a formal method for software development was first proposed in the 1970s as an implementation-independent specification technique for abstract data types [14], [15]. Over these years, it has been advanced to specify concurrent systems, state-based systems and software components based on solid foundations of the theories of behavioural algebras [16] and co-algebras [17]–[20]. In comparison with other formal approaches, algebraic specifications are at a very high level of abstraction. They are independent of any implementation details. Another attractive feature of algebraic specifications is that they can be used directly in automated software testing [21]–[25]. This feature is particularly important for service engineering, because when services compose together dynamically testing must be performed automatically on-the-fly.

In our previous work, in order to apply algebraic method to service-oriented software, we extended and combined the behavioural algebra and co-algebra techniques and revised the algebraic specification language CASOCC, which was originally designed for the specification of traditional software entities such as abstract data types, classes and components [25], [26]. Its revised version CASOCC-WS was applied to the formal specification of Big Web Services [2]. A tool that can automatically generate the signatures of algebraic specifications from WSDL description of Big Web Services was also reported. More recently, we have also applied CASOCC-WS to the formal specification of RESTful Web Services and developed a tool to perform syntax level consistency checking [27]. A case study with algebraic specification of a real industrial system GoGrid has been conducted [28]. Base on these work, a new algebraic formal specification language called SOFIA is proposed to improve the practical usability of algebraic specification languages [3].

Work has been reproted in the literature on transforming signatures of algebraic specifications into object-oriented class signatures [29]. They use the traditional algerbaic specification language, which allows the definition of several sorts in one specification unit. This makes the transformation much more complicated. For example, when transforming an operation into a method, it has to be classified to determine which class it belongs to. Our new specification language SOFIA enforces specifications to be well structured where only one sort is defined in one specification unit. Using SOFIA, the transformation of specifications into class signature could be much easier. Moreover, the transformation of specifications into domain ontology and ontological description of service semantics is more complicated, because in addition to recognise classes and the methods in the specifications, we also need to analyse various relationships between classes, individuals, etc.

B. Future work

We are pursuing towards a formal approach for specifying and testing service-oriented system. Currently, we are defining the formal semantics of the algebraic specification language SOFIA in order to lay a solid theoretical foundation of the proposed approach. We are also developing a tool that

uses specifications in SOFIA as input to perform automated testing and verification of web services. Another future work is to check the consistency of specification based on the ontological reasoning as well as equational logic inferences.

ACKNOWLEDGEMENT

The work reported in this paper is partially supported by EU FP7 project MONICA on Mobile Cloud Computing (Grant No.: PIRSES-GA-2011-295222), National Natural Science Foundation of Jiangsu Province, P.R.China (Grant No. BK2011022) and Jiangsu Qinglan Project.

REFERENCES

[1] GoGrid.com, "Gogrid website," http://www.gogrid.com/, Feb. 2013, last Access: Feb. 20, 2013.

[2] H. Zhu and B. Yu, "Algebraic specification of web services," in *Proc. of the 10th International Conference on Quality Software (QSIC 2010)*. IEEE CS Press, 2010, pp.457–464.

[3] H. Zhu, D. Liu, and I. Bayley, "Reference manual of the sofia algebraic specification language," Department of Computing and Communication Technologies, Oxford Brookes University, Technical Report TR-CCT-AFM-01-2013, Jan. 2013.

[4] H. Zhu, "A note on test oracles and semantics of algebraic specifications," in *Proceedings of the 3rd International Conference on Quality Software (QSIC'03)*. Dallas, USA: IEEE CS Press, Nov. 2003, pp. 91–98.

[5] T. Mossakowski, L. Schröder, M. Roggenbach, and H. Reichel, "Algebraic- coalgebraic specification in CoCasl," *J. Log. Algebr. Program.*, vol. 67, no. 1-2, pp. 146–197, 2006.

[6] M. Uschold and M. Gruninger, "Ontologies: Principles, methods, and applications," *Knowledge Engineering Re-view*, vol. 11, no. 2, pp. 93–155, 1996.

[7] D. Martin, et al., *OWL-S: Semantic Markup for Web Services*, member submission 22 ed., W3C, http://www.w3.org/Submission/OWL-S/, November 2004, last access: May 25, 2012.

[8] The OWL Serivice Coalition, *OWL-S 1.1 Release*, The OWL Serivice Coalition, Dec. 2008, available at http://www.ai.sri.com/daml/services/owl-s/1.2/.

[9] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. Koenig-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg, "Web service modeling ontology (WSMO)," W3C, Tech. Rep., June 2005, member submission.

[10] J. Bruijn, H. Lausen, A. Polleres, and D. Fensel, "The web service modelling language WSML: An overview," in *Proceedings of the 3rd European Semantic Web Conference*. Budva, Montenegro: Springer-Verlag, 2006, pp. 590–604.

[11] M. J. Hadley, "Web application description language (WADL)," Sun Microsystems Inc., CA, USA, Tech. Rep. SMLI TR-2006-153, March 2006.

[12] J. Kopecky, K. Gomadam, and T. Vitvar, "hRESTS: An HTML microformat for describing RESTful web services," in *Proc. of WI-IAT'08*. Sydney, Australia: IEEE/WIC/ACM., Dec. 2008, pp. 619–625.

[13] J. Lathem, K. Gomadam, and A. P. Sheth, "SA-REST and (S)mashups: Adding semantics to RESTful services," in *Proc. of ICSC'07*, 2007, pp. 469–476.

[14] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright, "Initial algebra semantics and continuous algebras," *Journal of ACM*, vol. 24, no. 1, pp. 68–95, 1977.

[15] H.-D. Ehrich, "On the theory of specification, implementation, and parametrization of abstract data types," *J. ACM*, vol. 29, no. 1, pp. 206–227, 1982.

[16] J.A. Goguen and G. Malcolm, "A hidden agenda," *Theorectical Computer Science*, vol. 245, no.1, pp.55–101, 2000.

[17] C. Cîrstea, "Coalgebra semantics for hidden algebra: Parameterised objects and inheritance," in *Recent Trends in Algebraic Development Techniques, 12th International Workshop (WADT'97)*, 1997, pp.174–189.

[18] J. M. Rutten, "Universal coalgebra: a theory of systems," *Theor. Comput. Sci.*, vol. 249, no. 1, pp. 3–80, 2000.

[19] C. Cîrstea, "A coalgebraic equational approach to specifying observational structures," *Theor. Comput. Sci.*, vol. 280, no. 1-2, pp. 35–68, 2002.

[20] F. Bonchi and U. Montanari, "A coalgebraic theory of reactive systems," *Electr. Notes Theor. Comput. Sci.*, vol. 209, pp. 201–215, 2008.

[21] M.-C. Gaudel and P. L. Gall, "Testing data types implementations from algebraic specifications," *CoRR*, vol. abs/0804.0970, 2008.

[22] H. Y. Chen, T. H. Tse, F. T. Chan, and T. Y. Chen, "In black and white: An integrated approach to class-level testing of object-oriented programs," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, no. 3, pp. 250–295, 1998.

[23] H. Y. Chen, T. H. Tse, and T. Y. Chen, "Taccle: a methodology for object-oriented software testing at the class and cluster levels," *ACM Trans. Softw. Eng. Methodol.*, vol. 10, no. 1, pp. 56–109, 2001.

[24] L. Kong, H. Zhu, and B. Zhou, "Automated testing EJB components based on algebraic specifications," in *COMPSAC (2)*, 2007, pp. 717–722.

[25] B. Yu, L. Kong, Y. Zhang, and H. Zhu, "Testing java components based on algebraic specifications," in *Proceedings of the First International Conference on Software Testing, Verification, and Validation (ICST 2008)*, Lillehammer, Norway, April 9-11 2008, pp. 190–199.

[26] L. Kong, H. Zhu, and B. Zhou, "Automated testing ejb components based on algebraic specifications," in *Proceedings of the 31th IEEE International Conference on Computer Software and Applications (COMPSAC'07)*, vol. 2. Beijing, China: IEEE CS Press, July 2007, pp. 717–722.

[27] D. Liu, H. Zhu, and I. Bayley, "Applying algebraic specification to cloud computing–a case study of infrastructure-as-a-service gogrid," in *Proceeding of The Seventh International Conference on Software Engineering Advances (ICSEA 2012)*, 2012, pp. 407–414.

[28] ——, "A case study on algebraic specification of cloud computing," in *Proc. of the 21st Enuromicro International Conference on Parallel, Distributed and network-Based Processing (PDP 2013)*, Queens University Belfast, Northern Ireland, Feb. 2013, p. (in press).

[29] B. Doell and W. Dosch, "Transforming functional signatures of algebraic specifications into object-oriented class signatures," in *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC 2005)*. IEEE CS, Dec. 2005, pp. 323–332.