Towards Text-Based Domain-Specific Modeling Language for Representational State Transfer Compliant Services

Agi Putra Kharisma  
Department of Informatics Engineering  
Universitas Brawijaya  
Malang, Indonesia  
agi@ub.ac.id

Mochamad Sukrisno Mardiyanto  
School of Electrical Engineering and Informatics  
Institut Teknologi Bandung  
Bandung, Indonesia  
sukrisno@informatika.org

ABSTRACT
This paper discusses the development of text-based domain-specific modeling language. The covered domain is representational state transfer (REST) architectural style. With domain-specific modeling language, we can develop software in a higher level of abstraction than general-purpose programming language. Domain analysis is performed to classify different kinds of resources into several resource types, namely resource, individual resource, projection resource, container resource, list resource, collection resource, transitional resource, and subresource. The design of modeling language is represented in metamodel and the implementation of modeling language is represented in EBNF-like grammar using Xtext domain-specific language (DSL) framework. In conclusion, we have successfully developed a domain-specific modeling language and its validators.

CCS CONCEPTS
• Software and its engineering  
Software notations and tools  
Context specific languages  
Domain-specific languages

KEYWORDS
Representational state transfer, Domain-specific modeling language, Code generator

1 Introduction
REST is an architectural style for hypermedia-based distributed applications [1]. REST is an architectural style with relatively high level of abstraction. Software engineers, architects, and developers are free to implement it in a concrete form without any specific standards. On the other hand, Remote Procedure Call (RPC) architecture, which is often used as an alternative to REST, has been equipped with various standards such as Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL). Because REST is not bound by specific standards, REST-based services are often seen as lightweight and become generally more popular than SOAP [2][3].

Modeling tools could help developing software with high maintainability. Currently, UML (Unified Modeling Language) is the most well-known graphical modeling tool for object-oriented software development. However, UML notation still has limitations in modeling web applications, so there is UML extension called Web Application Extension (WAE) [4]. UML is a general-purpose modeling language that can be used in various types of software, it is even called the "lingua franca" in software engineering, but in reality, the use of UML in industry is very limited [5].

There are a number of researches in web and web service modeling [6][7][8]. Some of the main goals in modeling is the integration of data aspect, functionality, behavior, and user interaction. In this context, UML is still considered to have not been able to achieve that goal [9]. Even further than that, there are considered conflicts between object-oriented concepts in UML and REST concepts [10]. REST-based services can be modeled with a conversation model [11]. Recent research has produced a
graphical modeling language called RESTalk that can model interactions (conversation) between clients and servers in REST-style more expressively [12], then in subsequent research RESTalk also provides a text-based modeling language as a complement [13].

In general, researches in modeling language are divided into two major groups, namely graphical modeling and textual modeling. Each has advantages and disadvantages, graphical modeling tools tend to be expensive, while textual modeling is relatively not popular. There is also research that attempts to combine the advantages of the two [14]. The proposed modeling language in this research uses text-based modeling approach. Some of the advantages of text-based modeling are information content, speed of creation, integration with multiple languages, speed and quality of formatting, independence of tools and platforms, and version control [15].

2 Modeling REST Domain

REST is the term used by Roy Fielding to name an architectural style for network-based software. REST consists of four constraints, namely client-server, stateless, cache, uniform interface, layered system, and code-on-demand [16]. REST services involve three main entities, namely clients, intermediaries, and the REST service provider itself called origin server as shown in Figure 1. Entities interact with each other by request and response mechanisms involving hypermedia. The hypermedia, apart from being the type of media representation of a resource, is the core control of the application state which is used as a basis for consumption and state manipulation of the resource concerned. The modeling language proposed in this research is only focused on REST services from the point of view of the origin server.

![Figure 1: Context of REST](image)

An REST-based application is developed using a resource-oriented paradigm. In this context, a resource is everything that can be accessed through the uniform interface. This research classifies resources into several resource types. The classification process refers to another research that make resource classification too [17][18][19][20]. The proposed resource type classification is shown in Figure 2.

![Figure 2: Proposed resource type classification in this research](image)

The following is a description of each resource type we proposed in Figure 2:

1. Resource
   Resource is an abstract concept used as a reference of other resource types. Resource has an identifier and is manipulated through a uniform interface.

2. Individual Resource
   Individual resource is a resource that represents a singular concept. Example: student, book.

3. Projection Resource
   Projection resource is a resource that only contains partial attributes of another resource. Example: student address, book title, and date of university establishment.

4. Container Resource
   Container resource is a resource that contains another resource. Example: index, control panel.

5. List Resource
   List Resource is the specialization of container resource that resembles the list. List resource has a unique feature (filter and paging). Example: search result.

6. Collection Resource
   Collection resource is a specialization of list resource with the ability to add new resources into a collection. Example: a collection of books, collection of students.

7. Transitional Resource
   Transitional resource is a resource that doesn’t represent the concept of singular or plural, but represents transitional or transformation oriented. Example: study plan approval, staff assignment.

8. Sub-resource
   Sub-resource is a resource that is part of another resource. Example: picture of the book cover.

After that, we model the relationship between resource and representation in a high level of abstraction as can be seen in Figure 3. The model in Figure 3 is one of the bases for making the modeling language metamodel that we propose.
Towards Text-Based Domain-Specific Modeling Language for Representational State Transfer Compliant Services

Figure 3: Relationship between resource and representation

Figure 3 shows REST architecture modeled in a simple way with focus on one of the REST constraints, namely resource manipulation through its representation. Representation consists of resource representation including hypermedia link and hypermedia control. Resource representation comes from the resource state that located in the server. Resource state consumed and manipulated by the client through hypermedia (hypermedia link and hypermedia control) that serves to control application state (navigation and workflow) and changing the resource state.

2 BeREST Modeling Language

In this paper, we proposed a modeling language called BeREST. BeREST is a text-based domain-specific modeling language for application development with REST architectural style. Figure 4 shows BeREST metamodel that is used as a basis for designing BeREST language. The metamodel in Figure 4 is created by referencing to metamodel for a model-driven web engineering method [21].

The implementation of BeREST language is represented in grammar on Xtext development environment (https://www.eclipse.org/Xtext/). The grammar syntax on Xtext is similar to the Extended Backus-Naur Form (EBNF). BeREST divides the domain into three types, namely Resource, DataType, and MediaType. DataType defines the data type (example: String, Integer, and Boolean). MediaType is the type of media used in resource representation (example: XML, JSON, and plain text). Resource is one of REST main concepts, defined with selecting the appropriate resource type as defined in ResourceType enumeration in Figure 4. In general, BeREST grammar showed in the code excerpt below:

```
DomainModel:
  elements+=Element*
;
Element:
  Resource | DataType | MediaType
;
DataType:
  'Datatype' name=ID
;
MediaType:
  'Mediatype' name=ID
;
Resource:
  type=ResourceType name=ID
  (':' projection=[Resource])?
  (cache ?='NoCache')?
  '{
    requests+=Request*
  }'
;
Request:
  'request' name=ID method=HTTPMethod
  path=GENERAL_URI_TEMPLATE '{
    (body '{
      body+=ResourceAttribute*
    }')?
    (response '{
      (media-type mediaType+=[MediaType]((',
        mediaType+=[MediaType])*)?)?
      (attributes attributes+=ResourceAttribute*)
      (navigations navigation+=Navigation*)
    }')?
  }'
;
```

With the excerpt of the grammar above, we also add additional validation to maintain the validity of the resulting model. For example, when modeling a service with a POST request to IndividualResource type, which is considered illegal, the error can be immediately visualized in the modeling editor as shown in Figure 5.
domain modeled by BeREST. We try to limit the BeREST language automatically generates executable code, we try to narrow the to generate create, read, update, delete (CRUD) applications on relational databases. Due to the different domains, we named the specific language BeRESTDB. BeRESTDB is basically BeREST with specialization for CRUD based application on the relational database. The architecture of BeRESTDB is shown in Figure 6. For the generic CRUD functionalities, this can be fully generated automatically. As the target language of the code generator, we chose the Sinatra (http://sinatrarb.com) based on the Ruby programming language because it is already in a relatively very high level of abstraction, making it easier and simpler to generate.

3 Code Generation on BeREST

Creating an executable modeling language in the REST domain is still a challenge today. This is because the domain in REST may be too large to be fully automated with a code generator. Many studies have been carried out, generally only producing semi-automatic language. This means that the code generated by the code generator cannot be executed immediately without modification, because in some parts it is still a stub method which is generally used to define business logic [22].

In order to produce a modeling language that fully automatically generates executable code, we try to narrow the domain modeled by BeREST. We try to limit the BeREST language to generate create, read, update, delete (CRUD) applications on relational databases. Due to the different domains, we named the specific language BeRESTDB. BeRESTDB is basically BeREST with specialization for CRUD based application on the relational database. The architecture of BeRESTDB is shown in Figure 6. For the generic CRUD functionalities, this can be fully generated automatically. As the target language of the code generator, we chose the Sinatra (http://sinatrarb.com) based on the Ruby programming language because it is already in a relatively very high level of abstraction, making it easier and simpler to generate.

4 Discussion

REST domains have a very broad scope. Modeling of the REST domain is done from architectural and behavioral aspects. From an architectural aspect, REST already has clear constraints. However, from the behavioral aspect, especially with regard to business logic, the constraints are very broad. Each application can have very different business logic, especially when comparing applications in different domains. For example, the difference of business logic between marketplace application and online learning application may very significant which makes it difficult to carry out general abstraction. So that until now, we have not found a method to produce a REST domain-specific modeling language that is fully executable according to model-driven software engineering (MDSE) principles.

5 Conclusion and Future Work

We have successfully tried to develop a text-based domain-specific modeling language for REST-based services by analyzing the REST domain, classifying its elements, creating grammars for modeling languages, and developing code generators to produce a fully executable modeling language. The challenge faced is when defining specific business logic, because it relates to the domain in the application. It is very difficult to generalize the various business logic at a high level of abstraction. So that in the future, the modeling language needs to be limited to a narrower domain so that it can be fully executable.

ACKNOWLEDGMENTS

We thank colleagues at Universitas Brawijaya and Institut Teknologi Bandung for their suggestions and input to improve this research.

REFERENCES

Towards Text-Based Domain-Specific Modeling Language for Representational State Transfer Compliant Services


