

# A Model Driven Framework for the Development of Adaptable REST SERVICES

Adil Kenzi<sup>a</sup> and Fadoua Yakine<sup>b</sup>

LISA Laboratory, Sidi Mohamed Ben Abdellah University, FEZ, Morocco

**Keywords:** REST, UML Profile, Restful Web Services, MDD/MDA, Adaptability, Service Modeling.

**Abstract:** REST (Representational State Transfer) is an architecture style for distributed, open, loosely coupled and decentralized hypermedia systems such as the Web. In the context of this architectural style, Restful Web services has gained significant attention in both academy and industry sectors. Restful Services may interact with several types of service requesters. Therefore, the key issue is how to deal with the challenge of adaptability of Restful Services. In this paper, we propose a framework for the development of adaptable REST services. The core building blocks of this framework is a Unified Modeling Language profile called RESTVSoaML, and its associated tool support RESTVSoaMLTool. RESTVSoaML aims the modeling of adaptable Restful Web services regardless of standards and implementation platforms. RESTVSoaMLTool is an MDD tool that enables the generation of code by using a model transformation language, from high level models defined with our profile RESTVSoaML. In particular, it permits the generation of the description of each RESTFUL service and its implementation.


## 1 INTRODUCTION


REST (Representational State Transfer) is an architecture style for distributed, open, loosely coupled and decentralized hypermedia systems such as the Web (Pautasso, Wilde and Alarcon, 2013). In the context of this architectural style, Restful Web services has gained significant attention in both academy and industry sectors in comparison to SOAP services. Many mainstream Service providers (e.g. Yahoo, Google, and Facebook) has adopted REST Services due to its simplicity of use.

Restful Web Services can be viewed as software services which are published on the Web. Such services probably interacts with several types of service requesters that have different needs and requirements. Therefore, the central problem is the design and development of highly adaptable/personalized Restful services. To tackle this problem, we put forward a model driven framework for the design and development of adaptable Restful services based application. Such a framework provides the necessary capabilities to model users' needs and requirements early in the

development lifecycle of service based applications and enables the automatic code generation from high level platform independent models. Indeed, several approaches have been proposed to take into account several aspects of service requesters needs such as the adaptation to their profiles (Chang and Kim, 2007), the management of their access rights (Hafner and Breu, 2008) (Fink, Koch and Oancea, 2003) or the adaptation to their contexts (device, location) (Bouguettaya and Yang, 2009). Nevertheless, to the best of our knowledge, there is no framework that enables the modeling of users' needs by separating their concerns early in the development process of service based application and permitting the automatic generation of code following the model driven development principles.

In this paper, we propose a model driven framework for the design and development of adaptable Restful Services. The core building blocks of this framework are: a UML profile called RESTVSoaML and A MDD tool. (1)RESTVSoaML (REST View based Service Oriented Architecture Modeling Language) is a UML profile enabling the design and specification of adaptable Restful

<sup>a</sup>  <https://orcid.org/0000-0002-5800-1968>

<sup>b</sup>  <https://orcid.org/0000-0001-9789-6146>

Services. Such UML profile defines a set of stereotypes that allows the representation of adaptable Restful Services in a high level of abstraction. The key element of the proposed profile is the REST multiview service defined as a first class modeling entity that allows the representation of the needs and requirements of users by separating their concerns early in the development lifecycle of Services based Applications. The REST multiview service as a new modeling entity provides, in addition to the simple interfaces, interfaces which have the characteristic of being flexible and adaptable to the different type of service requesters.

(2) The MDD tool associated to the profile called RESTVSoaMLTool allows the automatic generation of code from high level models following the MDA principles and standards. It permits also the deployment of each REST multiview service. In a first phase, RESTVSoaMLTool accepts in input platform independent models that describe the structure and functionalities of systems according to the actors interacting with each service and by using a model transformation language, it generates automatically the description of each REST multiview service (e.g. WADL) and its implementation (e.g., JAX-RS).

The rest of this paper is structured as follows: Section 2 presents our UML profile RESTVSoaML. In section 3, we illustrate the applicability of our profile on the basis of a motivating example. In Section 4, we present an overview of the RESTVSoaMLTool tool allowing both the edition and transformation of models and deployment services. Section 5 illustrates the steps to generate the WADL description. Section 6 presents some related works, and in Section 7 we give a conclusion and perspectives to our work.

## 2 A UML PROFILE FOR RESTFUL ADAPTABLE SERVICES

The objective of this section is to present the first component of our approach: the UML profile for adaptable restful services. To this end, we put forward the REST MULTIVIEW SERVICE CONCEPT. Finally, we describe our UML profile for the modeling of adaptable restful services.

### 2.1 Rest Multiview Service Concept

The concept of view is principally adopted in different computing fields such as Database Management System (Rafanelli, 2002), Workflow, Web Services (Fink, Koch and Oancea, 2003) (Maamar *et al.*, 2005) (Rademacher *et al.*, 2019) etc. In the context of this paper, we adopt the view concept as a means of functional separation of concerns by highlighting what is expected to a specific type of actor. Generally, the separation of concerns (Ossher and Tarr, 2001) helps in writing software that is modularized by concern; modeling concerns and their relationships and extracting concerns that are tangled with others.

We define the concept of REST multiview service as a first class modeling element that illustrates the user needs and requirements early in the development process of service based applications. The REST multiview service enables the capture of the various needs of service clients by separating their concerns. For each service consumer, the service must provide the required capabilities that correspond to the needs of users invoking the service. From an analysis/design viewpoint, the central problem therefore, is how to model the multidimensional aspect of the needs of the various actors interacting with the same service. Thus, a REST multiview service provides in addition to the simple interfaces that characterize the service, service interfaces which are able to describe the capabilities of services according to the profiles of users interacting with the same service. We call this type of interfaces a **RESTviewServiceInterface**. For each REST multiview service, we group semantically the various **RESTviewServiceInterface** in a set of packages. Each package called **RESTMVServiceInterface** is composed of a base interface (**RESTbaseServiceInterface**) and the set of view interfaces (**RESTviewServiceInterface**). The **RESTbaseServiceInterface** permits the representation of the functionalities of services required by all kinds of users. In contrast, the **RESTviewServiceInterface** permits the representation of the functionalities required by a specific kind of user. These functionalities are accessible only if the specific user is in interaction with the service. The **RESTviewServiceInterface** depends on the base interface in the sense where the functionalities of the **RESTbaseServiceInterface** are implicitly shared by all views interfaces.

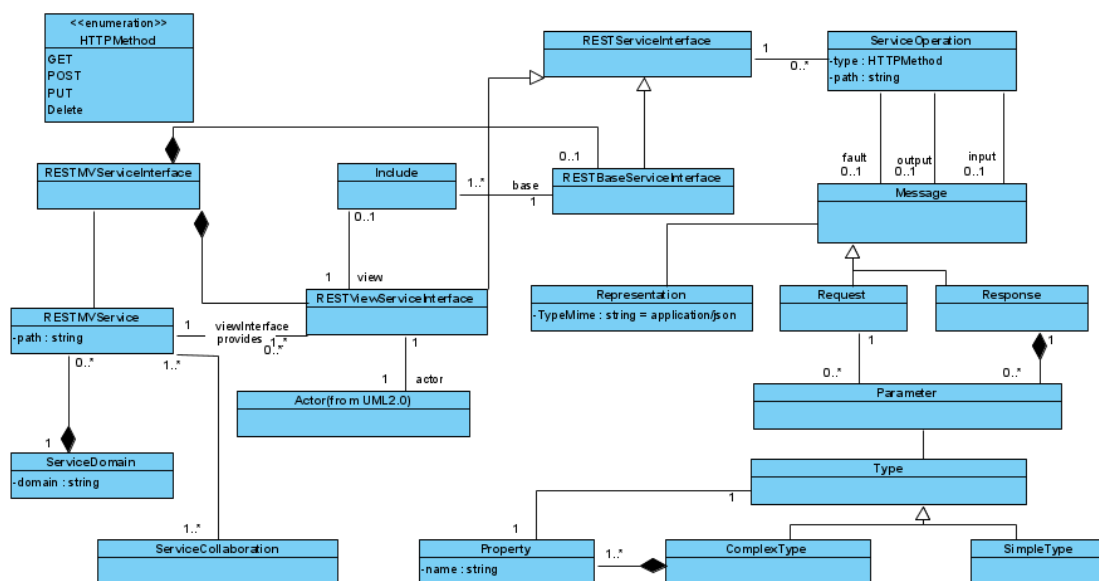


Figure 1: Excerpt of the RESTVSOAMLMetamodel.

## 2.2 A UML Profile for Adaptable Restful Services

The REST multiview service as a first class modeling entity plays a crucial role in the specification of adaptable service based applications. Indeed, in Service Computing the service is a fundamental element for the development of interoperable, evolvable, distributed information systems (Papazoglou, 2008). Each service oriented system is mainly composed of a set of services which are described, published, discovered and can be assembled in order to create more complex based systems and distributed applications in a cost and time effectiveness way.

Obviously, the REST multiview service constitutes the core building block of our approach, however it is insufficient to describe all aspects (structural and behavioral) of service oriented systems such as the information models associated with each service or the service collaboration which must specify the composition of REST multiview services to cope with the complex nature of users' needs that an atomic service does not deal with them. Hence, we must define other concepts and notations in order to describe the various aspects of an adaptable service oriented systems. Such concepts and notations are defined in the context of our UML profile **RESTVSoaML**.

**RESTVSoaML** allows the modeling and specification of adaptable service oriented systems. It defines a consistent set of stereotypes, its associated notations, constraints and semantics. In other words,

the profile defines a Domain Specific Language (DSL) providing designers the necessary capabilities to design an adaptable Service based application in high level of abstraction regardless of implementation platforms (dotNet, JEE, etc.) and standards (WADL, XML, etc.).

Figure 1 depicts the RESTVSoaML metamodel. Such metamodel plays an important role in the process of the automatic code generation, especially in the context of a Model Driven System Development (MDS). Generally, the profile defines in addition to the **RESTMultiview** service, the **RESTMVServiceInterface**, **RESTbaseServiceInterface**, **RESTviewServiceInterface** described in the previous section, the following stereotypes.

**Message:** The stereotype Message represents the structures of data exchanged between service and service clients. These informations consist of data passed as input or output of a service operation. The use of this stereotype may be optional since it is possibly to use the parameters of each service operation in order to define messages.

A message stereotype has a property to specify its assumed encoding form (text, xml, json, etc.). A message may not have operations, it may have public properties and associations to each other messages.

**Service Interface:** The stereotype service interface is a modeling entity that represents the functional capabilities of a given service as a set of operations provided to its service clients. The service interface in addition to the service implementation constitutes a key artifact of the concept service in SC.

In the case of web service technology as a Web based implementation of Service Computing, each service interface is described by means of a WSDL or WADL document.

**Service Domains:** The stereotype “Service Domains “ represents some logical or physical boundary of the system. In fact, the specificity of SOA is the integration of application both intra an inter-enterprises, so it is necessary to distinguish between the different collaborating domains.

**ServiceCollaboration:** The stereotype “ServiceCollaboration” allows the specification of a set of atomic services in order to create more complex added value service in a high level of abstraction.

### 3 DLS CASE STUDY: ABSTRACTION LAYERS

Our case study is a simplified version of a DLS (Distance Learning System). Our objective is to use this simplified version in order to validate and illustrate our approach. The DLS interacts mainly with three actors: Students, Teachers and Administrators. For each actor, the DLS must provide only the functionalities corresponding to its needs. It allows students to apply for courses, access to documentation (slides, web pages, text, etc.), make exercises, communicate with teachers, and take exams. The DLS provides for students runtime sequences, such as getting the next item in a sequence. DLS can be distributed over several sites, and is managed by a responsible whose job consists in: Student registration (registering students for available courses); Resource management (creating, updating and removing resources, their availability and their interactions) and Content management (creating, updating, organizing and publishing information resources). Teachers use the DLS in order to propose and update their own courses; plan learning experiences and units of work for delivery on or off line; access curriculum outcomes and record student assessments. They are in charge of writing exam subjects.

In order to develop an adaptable service oriented DLS, we have defined several abstraction layers to master the complexity associated to such systems. Globally, we have defined three abstraction layers: the business layers, the logical layer and the physical layer.

**The Business Layer:** The business layer is defined on the basis of a set of models in order to describe the business requirements. In this layer, we have used the formalism of use cases allowing the formalization of user requirements.

**The Logical Layer:** The objective of the logical layer is the description of the services which will compose the system at a high level of abstraction. In our approach, the logical layer is mainly composed of a collection of REST multiview services. In the context of our case study, we have identified some relevant DLS REST multiview services. The identification of these multiview services is based on the development process defined in (Kenzi *et al.*, 2009). Such development process allows the transformation of the business models into multiview services models.

The multiview service model of the DLS is composed mainly of the REST multiview services: **Registration, Course, Documentation and Exam**. Such services are multiview since they interact with many actors: Teacher, Student and Administrator. For example, Registration service provides the student with the required functionalities in order to register while it enables an administrator to manage the registration of students, to fix the registration fees to given courses and to make decisions about the registration of students. The **Course service** provides functionalities permitting students to apply for courses. They can have access to exercises, ask questions, consult projects, take quizzes and post messages to discussion forums relating to a specific course. It also enables Teacher to create courses, to answer to students’ questions, to propose exercises and to reply to messages of discussion forums. The administrator uses course service in order to fix course fees, to manage the calendar of courses, to affect courses responsible. The **Documentation service** allows Teacher to upload documents concerning specific course, to update/delete the content. It permits students to download the required documentation (pdf file, ppt files, audio/video, etc). The **Exam Service** enables students to take exams, to download exam subjects, to ask questions, to consult their marks. It also permits to Teacher to add exams, to answer students’ questions. Figure 2 illustrates examples of multiview services, specially, the **multiview service Course**. Each **Multiview Service** provides a set of **viewServiceInterface** and a **baseServiceInterface** that specifies the functionalities of the service required by all types of actors. The **viewServiceInterface** highlights the service capabilities required by a specific type of actor.

The physical layer consists of various Service based applications artifacts. It is primarily composed from the multiview service descriptions, multiview service implementations with a given programming language (e.g., Java, C#, etc), the BPEL code as well as various non-functional properties such as XACML policies for the management of access control.

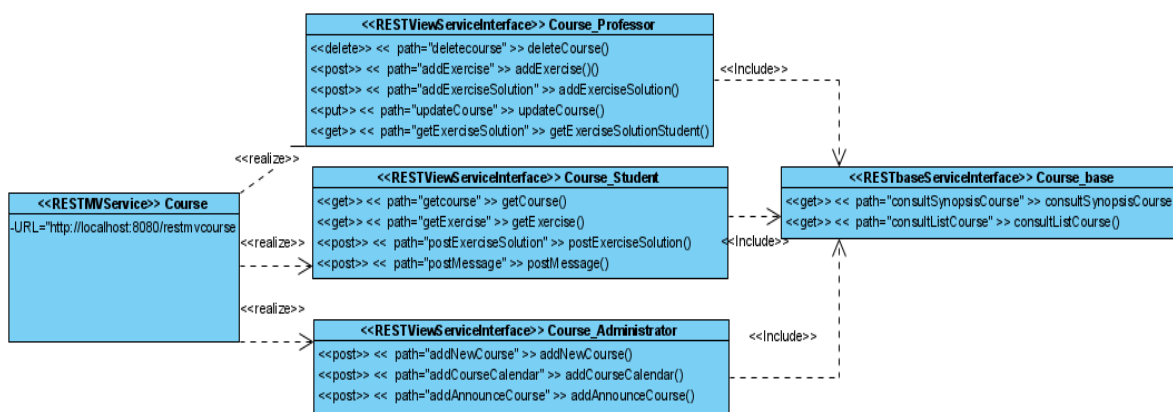


Figure 2: An example of REST multiview Service.

#### 4 RESTVSOAMLTOOL: A MODEL DRIVEN DEVELOPMENT TOOL FOR ADAPTABLE RESTFUL SERVICES

Our approach for developing adaptable service based applications is based on two components: the RESTVSOAML profile and its tool support called RESTVSoaMLTool. As illustrated in the previous sections, RESTVSoaML allows the description of adaptable SOA applications at high level of abstraction. The RESTVSoaMLTool aims the edition of models, the automatic generation of code from these models as well as the deployment of the REST multiview service descriptions following the web service reference architecture. For these purpose, RESTVSoaMLTool is composed of three main modules, as depicted in figure 3:

1. The Editor module that enables the edition of RESTVSoaML models on the basis of the EMF ([www.eclipse.org/emf/](http://www.eclipse.org/emf/)) tools such as Ecore.
2. The transformation module allows the generation of code from the edited models. It is based on two complementary transformations targeting two specific platforms. Each transformation is carried out on two steps: Model to Model transformation and Model to Code transformation. The first transformation aims the generation of the REST multiview service description that conforms to the WADL. The second transformation aims the generation of the service implementation according to a particular implementation platform (dotNet, JEE, etc.). In our approach, we have chosen JAX-RS as an implementation platform. Indeed, JAX-R is used to implement

Restful web services in JEE platforms. To achieve the transformation targeting a JAX-RS implementation platform, we firstly define the JAX-RS metamodel. Secondly, we specify the mapping of the RESTVSoaML metamodel as source metamodel to the JAX-RS metamodel as a target metamodel by identifying the equivalent elements. Thirdly, we define the transformation rules which implement the equivalences between the source and target metamodel elements. Finally, we define additional transformations in order to generate the java code as an implementation of the multiviews service.

3. The deployment module allows the deployment of each generated REST in a given application server such as Apache Tomcat or GlassFish.

#### 5 GENERATION OF MVWADL DOCUMENT

In this section, we present our extension to WADL called MVWADL for the description of REST multiview Service and its metamodel. Also, we illustrate how to generate MVWADL documents

##### 5.1 MVWADL Metamodel

WADL (Hadley, 2006) is an XML based language for the description of RESTfull Web services. WADL plays the same role as WSDL for SOAP Web Services. WADL describes the interfaces of restful and the relationships between them. Moreover, It permits the definition of the media types used for the representation of the HTTP request as well as the HTTP response. However, the WADL specification does not take into account the profile of users that

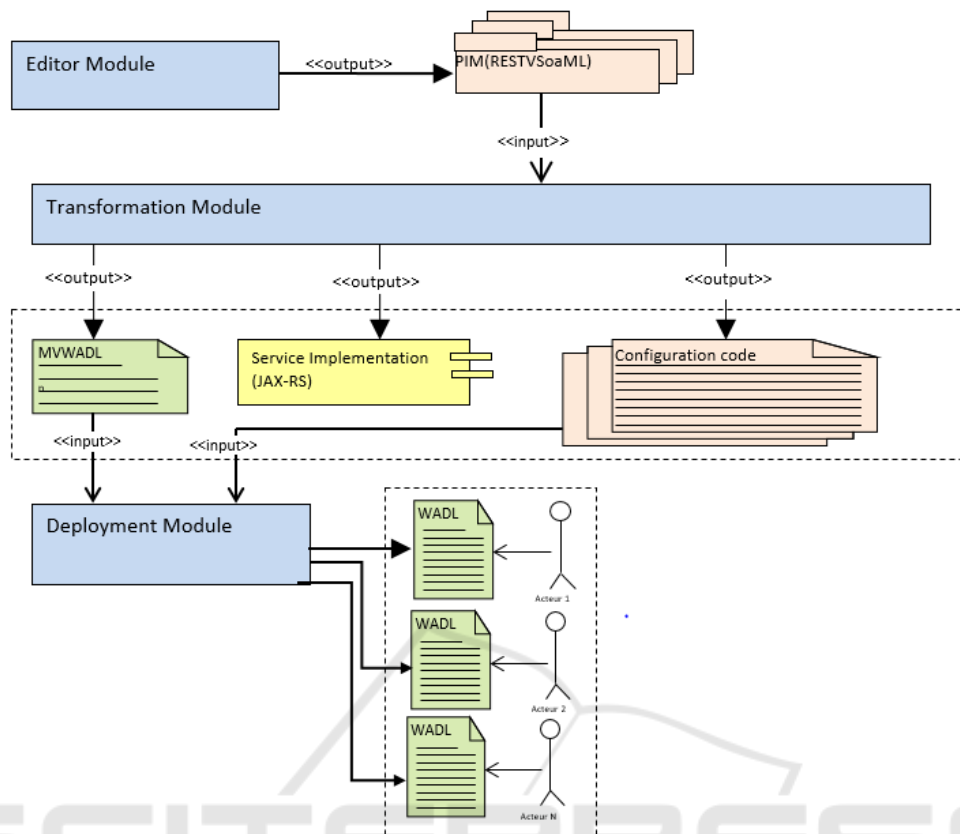


Figure 3: Main modules of RESTVSoaMLTool.

interact with the service. Indeed, two service requesters with different profiles can have the same WADL. To deal with this problem, we put forward a lightweight extension of the WADL specification called MVWADL (MultiView Web Application Description language) in order to describe the REST multiview service. The purpose of this extension is twofold. Firstly, MVWADL describes in a single XML file all REST service interfaces both simple and multiview interfaces. Secondly, it permits the adaptation of this description taking into account the profile (i.e the role) of the user invoking the service by providing a WADL description adapted to the users' profiles. We have also defined the Multiview WADL Meta-model (cf. Figure 4) as an extension of WADL meta-model in order to take into account the profile of the user interacting with the service. This extension is carried out by means of an element called "actor". Such an element permits the definition of the profile interacting with the service. It is associated with different element of WADL that must be adapted to users' profiles such as **Resource**, **grammars**, etc.

In addition to the **actor** element, the MVWADL metamodel is composed from the following elements:

- **application** is the root element in a WADL document. It defines global information about services, especially references to the schema definition.
- **Resources** is an element that contains all the resources provided by a RESTful Web Services. This element defines a base attribute which identifies the common path used by all resources in the RESTful Web Services.
- **grammars**: This element includes the definition of data formats using the **include** element, which references the data format with the **href** attribute.
- **resource** is child element of the **resources** element used for defining a collection of resources. Each **resource** is identified by unique URI and can also comprise the element **param** to identify the path parameters contained in the request.
- **method**: The element **resource** may be associated with one or more **method** element. The element **method** defines the HTTP method (e.g. get, put, post, delete). Each **method** has as an input a **request** and produces in **output** a response.

- **request:** A **request** can have one or more child elements for example:
  - o **representation:** This element defines the internet media type (JSON, XML, HTML, text, ...) for the body of the request
  - o **param:** This element defines the query or header parameter contained in the request
- **response:** The element **response** defines the result of the call of a **method** on a **resource**. The **response** element contains the optional status code contained in the HTTP response and can also contains the data in given produced by a HTTP method, especially in the case of a GET method. A response can have several child elements such as:
  - o **representation:** This element describes the internet media type for the response body returned by the execution of a restful web service method.
  - o **param:** This element defines the HTTP header parameters contained in the response.

transformation. In M2M transformation, the input and the output parameters of the transformation are models while in M2T transformation, the output is Text String. In our approach, the transformation is carried out on two phases: the mapping phase and the definition of transformation rules phase. In the mapping phase, we have determined the elements of the source metamodel which are equivalent to the elements of the target metamodel. Table I shows a possible mapping from the RESTVSoaML metamodel to the MVWADL metamodel. In this table, the first column contains the elements of the MVWADL metamodel. The second column presents its equivalents elements from RESTVSoaML metamodel.

Table 1: Mapping between WADL element and RESTVSoaML.

| MVWADL element | RESTVsoaML   |
|----------------|--|
| Application    | RESTMVservice  |
| Grammars       | Type   |
| Resource       | RESTServiceInterface, RESTviewServiceInterface, RESTBaseServiceInterface |
| Method         | ServiceOperation   |
| Request        | Request  |

## 5.2 Transformation Module: The Generation of MVWADL Document

Model transformation (Brambilla, Cabot and Wimmer, 2017) is a key technique in model driven Engineering. We distinguish between two types of model Transformation: Model to model (M2M) transformation and Model to Text (M2T)

The mapping specification allows the specification of the equivalent elements between the source and target metamodels elements. The second phase of the transformation process consists on the definition of the transformation rules using a model transformation language.

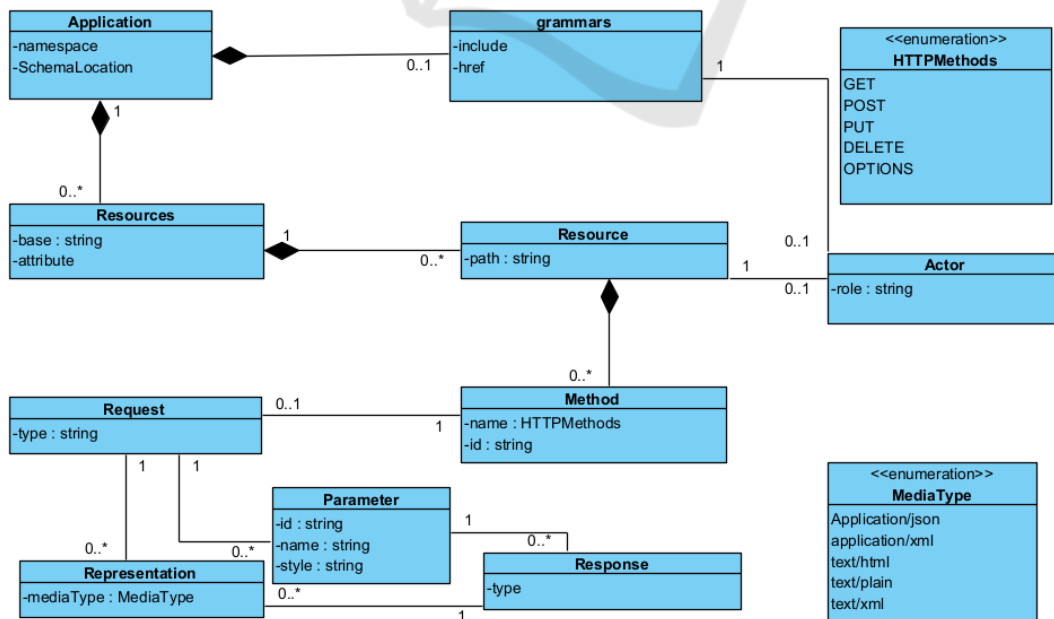


Figure 4: The MVWADL metamodel.

In the context of our Framework, we are using ATL (Atlas Transformation Language) language to implement a set of transformation rules that allows the M2M and M2T transformations in order to generate MVWADL documents.

After the generation of MVWADL, the deployment module can also produce a standard WADL document for each actor interacting with the multiview REST service as depicted in figure 3.

## 6 RELATED WORK

Model driven web service development has been an active area of research in software engineering. Thus, Gronmo et al. (Gronmo *et al.*, 2004), Bezivin et al. (Bezivin *et al.*, 2004) and Yu et al. (Yu *et al.*, 2007) propose several approaches in the MDA context for the development of web service systems. These approaches target the automatic generation of the code from a given PIM. They are based essentially on the elaboration of PIM on the basis of a high level modeling language (UML, EDOC). Then, this PIM will be transformed in a web service platform by using a transformation model language (ATL, QVT) or a specific tool such as presented by Gronmo et al. (Gronmo *et al.*, 2004). Douibi et al. (Ed-Douibi *et al.*, 2016) put forward an approach called EMF REST, using EMF data models and allows the generation of web applications according to the REST principles. Haupt et al. (Haupt *et al.*, 2014) propose a multilayered metamodel for restful applications. The authors illustrate an implementation of their approach on the basis of a metamodel and a method. In the same context, Terzic et al. (Terzić *et al.*, 2018) present the MicroBuildertool for the specification of REST services. MicroBuilder is based mainly on two modules: the MicroDSL and the MicroGen. MicroDSL defines a DSL for the specification of REST services. MicroGen allows the generation of executable programs from high levels models defined by MicroDSL. (Rossi, 2016) put forward a model-driven approach to REST API development; two main phases are identified : (i) the definition of specific UML profile for modeling REST API and (ii) a model transformation that uses RAML as an intermediate notation that can be exploited to automatically generate documentation as well as code for various languages/platforms. In the same objective, (Zolotas *et al.*, 2017) present a model driven engineering engine that allows fast design and implementation of restful Web Services. The proposed approach allows developers to design their envisioned system on the basis of software

requirements in multimodal format. The input model for the MDE engine is created from textual requirements and graphical storyboards. Then, the MDE engine applies model to model transformations (CIM to PIM, PIM to PSM, PSM to code) in order to generate a restful ready to deploy web service. More recently, (Alulema *et al.*, 2020) propose a model-driven engineering approach for the service integration of IoT systems. In particular, the authors developed a tool for modelling IoT nodes with communication and integration capabilities. It consists of a graphic editor and a code generator, which generates software artefacts (IOT nodes, RESTful Web service, etc.).

Despite, several MDA approaches have been proposed in order to user into account. Thus, (Hafner and Breu, 2008) presented a model driven security engineering framework focusing on trust management issues. This framework permits the generation of platform specific security models such as XACML models via a set of transformation rules.

(D'Ambrogio, 2006) defines model driven approach for the generation of Q-WSDL(Qos-enabled WSDL) specification.

(Ortiz and Hernandez, 2006) propose an approach that combines between the SCA(Service Component Architecture), the Aspect Oriented Programming and the MDA in order to deal with the extra-functional properties such as encryption, logging, real time, etc.).

The main difference between these approaches and ours, resides in the PIM and PSM levels. The PIM which we put forward is essentially composed of REST multiview services that represent the users' needs and requirements.

## 7 CONCLUSIONS

In this paper, we have proposed a model driven framework for the design and development of highly adaptable Restful services. therefore, we have firstly presented a UML profile that enables the specification of Adaptable REST Services. Such UML profile defines a set of stereotypes allowing the representation of adaptable Service based applications in a high level of abstraction. The key element of the proposed profile is the REST multiview service defined as a first class modeling entity that allows the representation of the needs and requirements of users by separating their concerns early in the development lifecycle of service based Applications.



The second contribution of our approach is the tool RESTVSoaMLTool as a MDD tool associated with the profile enabling the automatic generation of code (e.g., WADL description) from high level models according to the MDA principles and standards.

## REFERENCES

- Alulema, D. *et al.* (2020) 'A model-driven engineering approach for the service integration of IoT systems', *Cluster Computing*, 23(3), pp. 1937–1954.
- Bezivin, J. *et al.* (2004) 'Applying MDA approach for web service platform', in *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004.*, pp. 58–70.
- Bouguettaya, A. and Yang, X. (2009) *Access to Mobile Services*.
- Brambilla, M., Cabot, J. and Wimmer, M. (2017) 'Model-driven software engineering in practice', *Synthesis lectures on software engineering*, 3(1), pp. 1–207.
- Chang, S. H. and Kim, S. D. (2007) 'A service-oriented analysis and design approach to developing adaptable services', in *IEEE International Conference on Services Computing (SCC 2007)*, pp. 204–211.
- D'Ambrogio, A. (2006) 'A model-driven wsdl extension for describing the qos of web services', in *2006 IEEE International Conference on Web Services (ICWS'06)*, pp. 789–796.
- Ed-Douibi, H. *et al.* (2016) 'EMF-REST: generation of RESTful APIs from models', in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 1446–1453.
- Fink, T., Koch, M. and Oancea, C. (2003) 'Specification and enforcement of access control in heterogeneous distributed applications', in *International Conference on Web Services*, pp. 88–100.
- Gronmo, R. *et al.* (2004) 'Model-driven web service development', *International Journal of web Services Research (IJWSR)*, 1(4), pp. 1–13.
- Hadley, M. J. (2006) 'Web application description language (WADL)'. Sun Microsystems, Inc.
- Hafner, M. and Breu, R. (2008) *Security engineering for service-oriented architectures*. Springer Science & Business Media.
- Haupt, F. *et al.* (2014) 'A model-driven approach for REST compliant services', in *2014 IEEE International Conference on Web Services*, pp. 129–136.
- Kenzi, A. *et al.* (2009) 'A model driven framework for multiview service oriented system development', in Aboulhamid, E. M. and Sevillano, J. L. (eds) *The 7th {IEEE/ACS} International Conference on Computer Systems and Applications, {AICCSA} 2009, Rabat, Morocco, May 10-13, 2009*. {IEEE} Computer Society, pp. 404–411. doi: 10.1109/AICCSA.2009.5069357.
- Maamar, Z. *et al.* (2005) 'Views in composite web services', *IEEE internet computing*, 9(4), pp. 79–84.
- Ortiz, G. and Hernandez, J. (2006) 'Toward UML profiles for web services and their extra-functional properties', in *2006 IEEE International Conference on Web Services (ICWS'06)*, pp. 889–892.
- Ossher, H. and Tarr, P. (2001) 'Using multidimensional separation of concerns to (re) shape evolving software', *Communications of the ACM*, 44(10), pp. 43–50.
- Papazoglou, M. (2008) *Web services: principles and technology*. Pearson Education.
- Pautasso, C., Wilde, E. and Alarcon, R. (2013) *REST: advanced research topics and practical applications*. Springer.
- Rademacher, F. *et al.* (2019) 'Specific Model-Driven Microservice Development with Interlinked Modeling Languages', in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 57–5709.
- Rafanelli, M. (2002) *Multidimensional Databases: Problems and Solutions: Problems and Solutions*. IGI Global.
- Rossi, D. (2016) 'UML-based Model-Driven REST API Development.', in *WEBIST (1)*, pp. 194–201.
- Terzić, B. *et al.* (2018) 'Development and evaluation of MicroBuilder: a Model-Driven tool for the specification of REST Microservice Software Architectures', *Enterprise Information Systems*, 12(8–9), pp. 1034–1057.
- Yu, X. *et al.* (2007) 'A model-driven development framework for enterprise Web services', *Information Systems Frontiers*, 9(4), pp. 391–409.
- Zolotas, C. *et al.* (2017) 'From requirements to source code: a Model-Driven Engineering approach for RESTful web services', *Automated Software Engineering*, 24(4), pp. 791–838.