Modeling Web Services with RoaML

Cătălin STRÎMBEI, Georgiana OLARU (AVRAM)
Al. I. Cuza University of Iasi
linus@uaic.ro, geo.olaru@gmail.com

In this paper we intend to argue whether a new service-based modeling approach could be technologically feasible, desirable by architects and developers and viable as new MDA platform. This new approach we call it RoaML as a "step-brother (or sister)" of already established SoaML initiative. We will debate whether RoaML can be a suitable modeling language for applications based on resource oriented architecture.

Keywords: *Web services, RESTful services, Service Oriented Architecture, ROA, SOA*

1 Introduction-Web Service-Based Architectures and UML Modelling

Originally intended to be a simple way of sharing and linking documents via the Internet, the Web has evolved into one of the most complex and successful technologies available nowadays, a transformation that has taken place in a short period of time. Consequently, it captured the attention of researchers that sought to investigate the recipe of its success and to extend the applicability of its fundamentals to other kinds of software applications as well.

Software architecture is an abstraction of the run-time elements of a software system. It is defined by the configuration of its elements – components, connectors, and data – constrained in their function and relationships in order to achieve a desired set of architectural properties (e.g., reliability, scalability, extensibility, reusability). Currently, two architectural styles are dominant: Service Oriented Architecture (SOA) and Resource Oriented Architecture (ROA).

The SOA and ROA architectural design patterns and the corresponding distributed programming paradigms provide a conceptual methodology and development tools for creating distributed architectures. Distributed architectures consist of components that clients as well as other components can access through the network via an interface and the interaction mechanisms the architecture defines; in the cases of ROA and SOA such distributed components will be named respectively resources and services.

With the emergence of SOA, a new UML specification was needed in order to cover the needs of designing services - SoaML. In this article we will argue whether another initiative more appropriate for the specific needs of modeling ROA applications.

1.1 SoaML Framework

Service Oriented Architecture (SOA) is the paradigm for the development of software systems based on the concept of service. A development method based on the SOA paradigm requires some notations to present services, their interfaces and the way they are built, including the case where they are built from other services, the architecture of a system in terms of services and the way they are orchestrated [6]. The SoaML specification defines a UML profile with a metamodel that extends UML to support the range of modeling requirements for SOA, including the specification of systems of services, the specification of individual service interfaces, and the specification of service implementations. The SoaML metamodel extends the UML metamodel to support an explicit service modeling in distributed environments. This extension aims to support different service modeling scenarios such as single service description, service-oriented architecture modeling, or service contract definition. This is done in such a way as to support the automatic generation of derived artifacts following the approach of Model Driven Architecture [9].

The OMG SoaML specification also introduces the concept of services architecture to model how a group of participants that interact through services provided and used to accomplish a result. According to SoaML, a service is an offer of value to a service consumer (a simple client or another service) through a well-defined interface that could be available to a community (which may be the general public). For SoaML, a service architecture is made by a group of participants providing and consuming services at specific service points [1]. The goals of SoaML are to support the activities of service modeling and design and to fit into an overall model-driven development approach, supporting SOA from both a business and an IT perspective.

1.2 IBM Rational Rose Framework: REST Service Model

Concerning the modeling of SOA requirements we have a well-defined standard (SoaML). Regarding the modeling of ROA requirements, we don't have any specific official guidelines. One attempt to customize the modeling and design for RESTful Web Services comes from IBM that has included in version 8.0.3 of Rational Software Architect Version a template for REST modeling [12]. This template proposes a set of elements for class and sequence diagrams like Resource Class, Path Dependency, GET Operation, PUT Operation, POST Operation, HEAD Operation and Delete Operation. These elements allow some kind of basic resource modeling but they don't offer other guidelines on how to model more complex architectures and we think that it does not represent a comprehensive modeling approach regarding ROA domain.

2 ROA vs. SOA

The traditional conceptual model of service-oriented architectures, or the service-oriented paradigm, seems like one evolutionary way of distributed computing programming. As Object Orientation paradigm has "naturally" evolved from procedural programming and modular development, challenged by distributed computing models based on RPC, T. Earl argues that service orientation evolved from object orientation, challenged by new distributed computing and integration models BPM, EAI and, finally, by web services standardized such as SOAP initiatives [3].

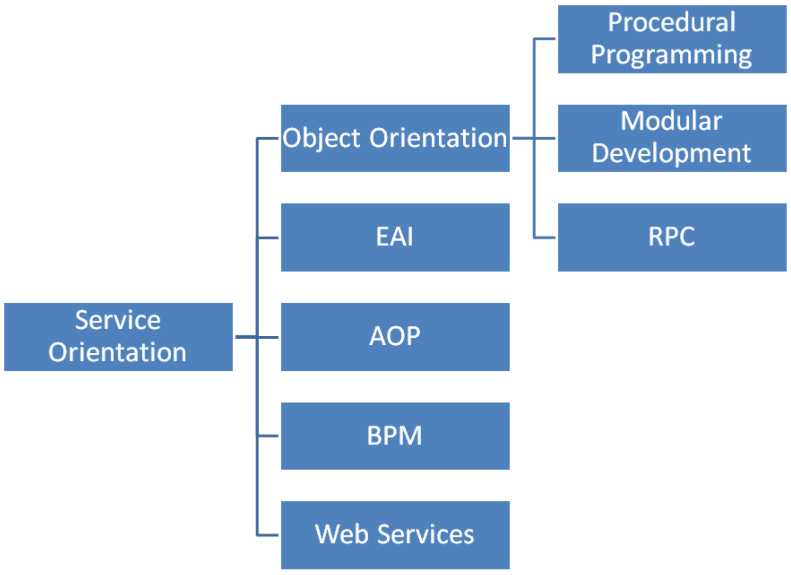


Fig. 1. Service Architecture Paradigm Source Origins

Our question is: REST oriented conceptual model could be the next evolutionary step of service-like architectures? Maybe … or maybe not? The main advantage of SOA over ROA is the more mature tool support, type-safety of XML requests. Conversely, the main advantage of ROA is the ease of implementation, the agility of the design, and a lighter approach for business perspective. Thus, REST services differ from older

SOA(P) services taking into consideration (at least) two perspectives:

- the degree of sophistication: REST “philosophy” assumes to simplify web service “protocol” as much as possible;
- the overemphasis on the basic and self-defining Rest principle of HATEOAS (Hypermedia as the Engine of Application State), something like “if there is not HATEOAS then there is no Rest”.

Taking into consideration the service design principles stated by T. Earl [3], [4], there are some subtle conceptual differences between SOA(P) and REST related to:

- service contracts, concerning standardization and design: RESTful requires no formal contract specification, although REST API standardization and versioning is promoted as best practice. In fact, RESTful proponents sustain very fluid REST APIs so that there is no “official” dependency to a formal service interface specification, REST API documentations are desirable but not to formally interfere with REST system architecture. There are some specialized tools to generate REST API docs, like Swagger, but there is no WSDL-like document implied. A service contract (like a WSDL document) looks like static typing (from programming languages) of the service providing component. As opposite, REST approach favor a very dynamic approach, e.g. service operations could be specified in service resource instance representation as dynamic-links and not through an extra meta-specification document;
- service coupling, concerning intra-service and consumer dependencies: any service-based architecture (or any kind of distributed component architecture) must have the attribute of interoperability. In this context, coupling refers to the way of managing service (component) interdependencies: there is an abstract level that could be understood as service structural relationships and there is a runtime level where those structural relationships became live (synchronous or asynchronous) connections. On the other side, REST services favor a more dynamic approach where link-lists, formatted using conventional standards like HAL, JSON-LD, Collection+JSON or SIREN as they are discussed in [10], could be generated specifically for each resource instance, unlike the more rigid approach of SOAP, where relationships are mostly endpoints statically defined in WSDL definitions;
- service discoverability, concerning interpretability and communication: SOA(P) promoters developed a sophisticated standard in this regard - UDDI, but, as REST means “simplification”, REST supporters considered that there is no need for a “middleman” like a “service registry”, the REST /service is a URL, and the URL must be formatted to be self-explained;
- service composability, concerning composition member design and complex compositions: SOA(P) approach, in the same line with service coupling principle, favor a contract-first pattern, as it is coined by R. Daigneau [7]. That means a static way to define composite services through WSDL specific documents, a declarative workflow spread in industry within the form of BPEL Orchestrators. On the other side, REST approach favor a more dynamic way to compose web services based on URL-links (as de facto relationships) close to point-to-point composability [2]. The are some critics to this kind of linking services which claim that point-to-point should not be considered as a kind of composability taking into consideration the that business logic is encapsulated in service implementation.

Therefore “traditional” SOA design principles are not quite entirely appropriate for RESTful-web services too, consequently ROA architectures might need different service design solutions.

3 Zero-Based Approach of RoaML

In our opinion, a radical innovative approach assumes a “zero-based approach” - meaning that it will not continue or refine an existing conceptual metamodel as “UML for Rest” or

"UML for ROA" or even "UML for SOA"... Our goal is to preserve simplicity declared by REST framework "founders" and theorists, but the "great compromise" is how to maintain consistency in the same time. In the following we rather propose some guiding principles for a business-oriented REST/ROA metamodel, and not a complete UML profile or framework for REST oriented architectures.

3.1 Metamodels

Our proposal takes into consideration an approach based on three delimited profiles, as in Figure 2:

- one focused on the application domain modeling, in fact the business side of the systems, that we named DDD Metamodel to invoke "Domain Design Driven" principle to model software components for business[8];
- one more substantial and consistent, named REST Metamodel, focused on modeling application services or components using RESTful principles. This metamodel also covers some elements dedicated to architectural modeling of more complex application systems from REST services, into a sub-profile named ROA Metamodel;
- last one, the REST Domain Metamodel, has the integrator role, so that metamodeling elements from Rest Metamodel (and ROA sub-metamodel) to be tailored for business specific needs.

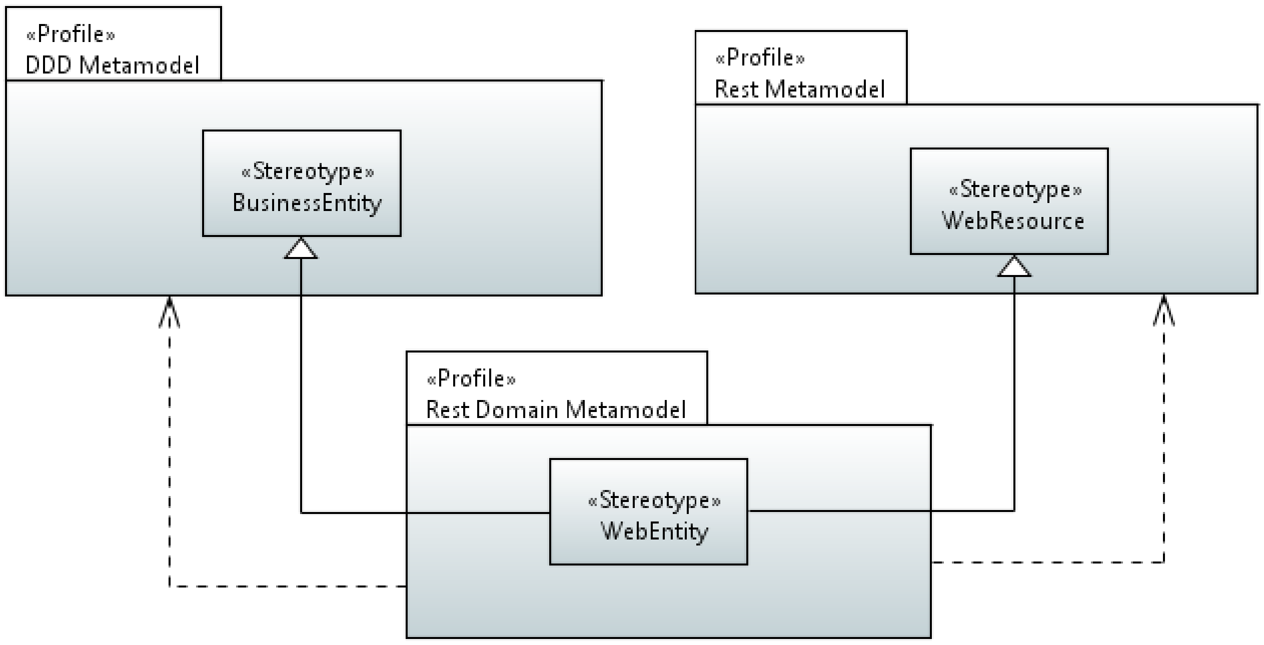


Fig. 2. RoaML Metamodels and their relationships

3.2 From REST Metamodel to ROA Metamodel

The core REST metamodel, makes a distinction between REST resources (marked with WebResource stereotype) and the REST services, as their producers, see Figure 3.

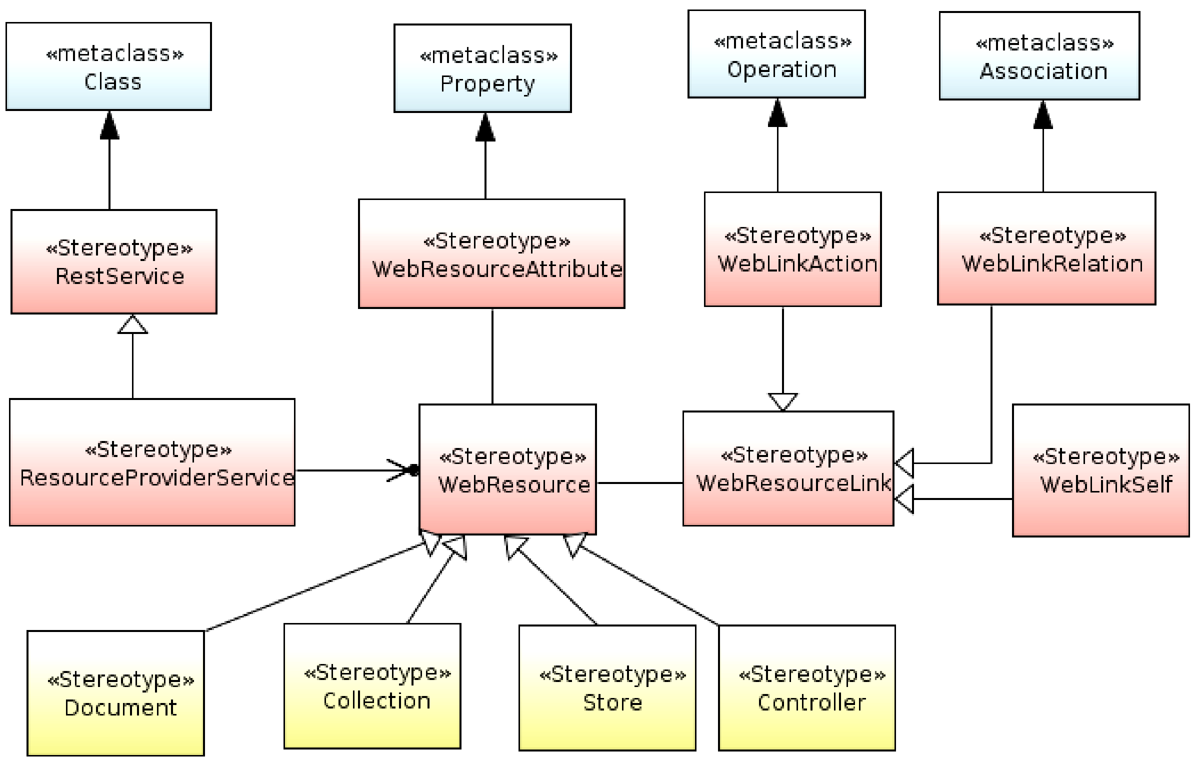


Fig. 3. Core REST Metamodel stereotypes and their relationships

Also this metamodel assumes the existence of resource descriptions where links have a fundamental role (considering HATEOAS principles). These hyperlinks could identify the resource (WebLinkSelf), could represent structural relationships with other resources (WebLinkRelation) or could signify an action endpoint concerning resource itself (other than standard actions based on GET, POST, PUT, DELETE requests from HTTP protocol). Another important distinction refers to resource archetypes [8] represented by a set of specialized WebResource stereotypes: Document, Collection of documents, Store and Controller.

A critical aspect of any component-based model concerns also the way to assemble individual components in complex systems. In order to address these issues in the context of REST services based architecture, we propose a meta-extension to the REST core-metamodel in the form of a ROA (sub)metamodel (shown in figure 4) based on the distinction between REST service and REST resource, that will take into consideration: (i) resource binding using relation-links; (ii)service binding using service-links guided by the link-relations from core resource model.

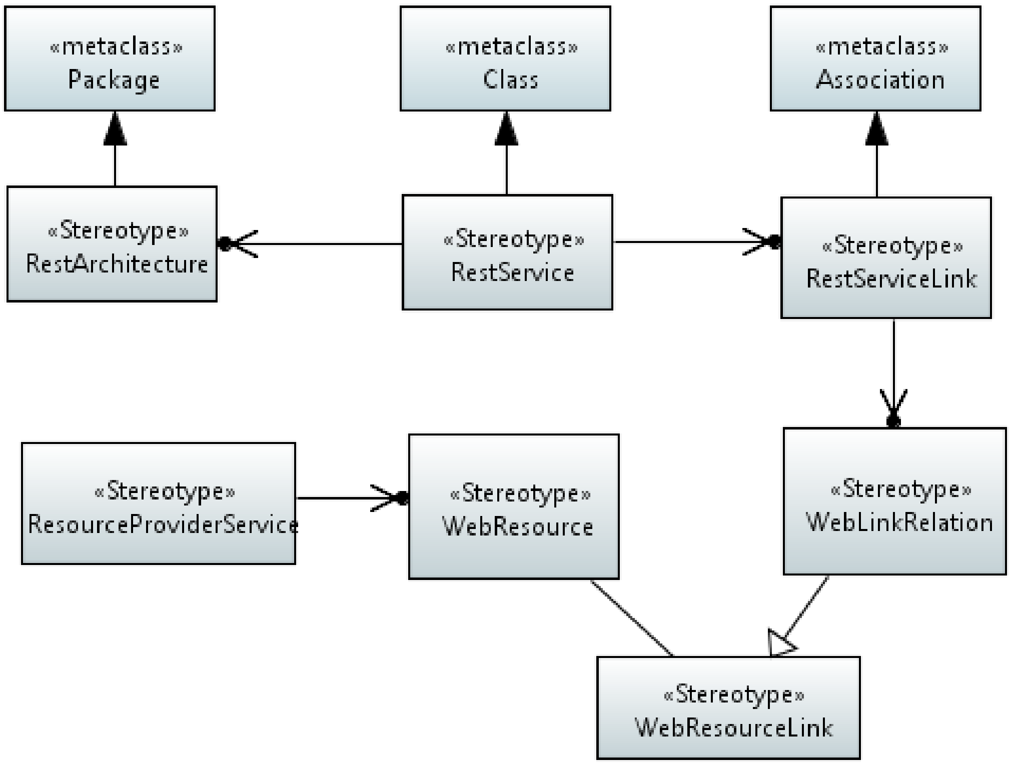

«metaclass» Package
«metaclass» Class
«metaclass» Association
«Stereotype» RestArchitecture
«Stereotype» RestService
«Stereotype» RestServiceLink
«Stereotype» ResourceProviderService
«Stereotype» WebResource
«Stereotype» WebLinkRelation
«Stereotype» WebResourceLink

Fig. 4. An architectural metamodel for ROA based systems

3.3 REST Domain Metamodel

Finally, the REST Domain metamodel (see figure 5) is centered on business WebEntity concept that tries to combine a fundamental concept from business metamodel with the REST WebResource from REST architectural model.

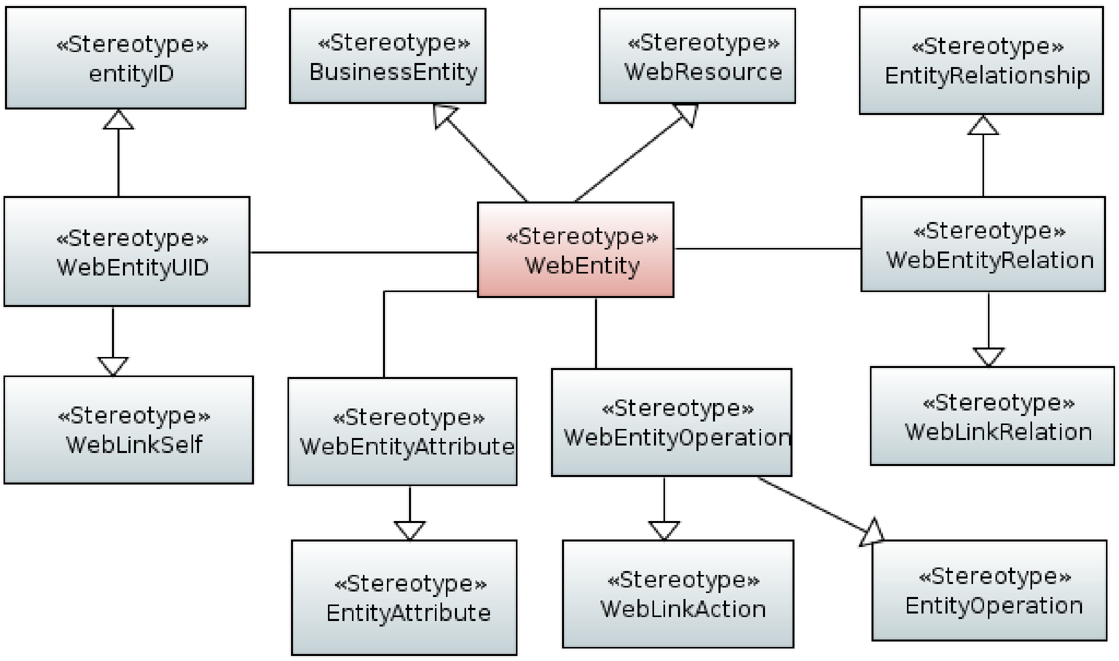

«Stereotype» entityID
«Stereotype» BusinessEntity
«Stereotype» WebResource
«Stereotype» EntityRelationship
«Stereotype» WebEntityUID
«Stereotype» WebEntity
«Stereotype» WebEntityRelation
«Stereotype» WebLinkSelf
«Stereotype» WebEntityAttribute
«Stereotype» WebEntityOperation
«Stereotype» WebLinkRelation
«Stereotype» EntityAttribute
«Stereotype» WebLinkAction
«Stereotype» EntityOperation

Fig. 5. An architectural metamodel for Rest service based systems

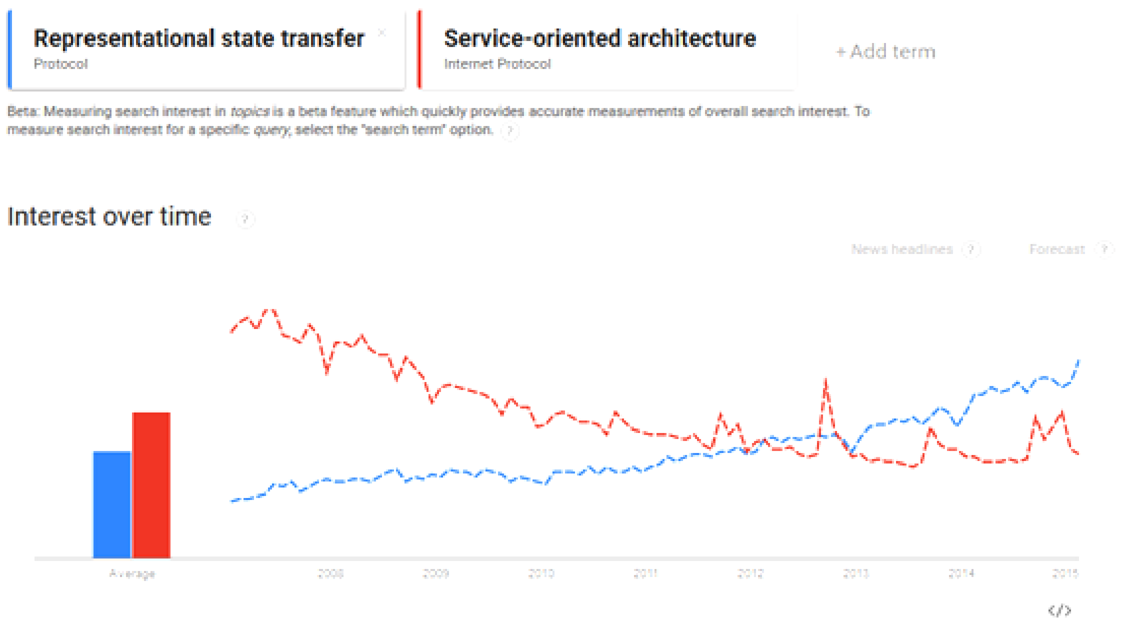
The WebEntity description will use at least three other meta-elements:

- entityUID for identity purposes, but in the form of self-links;
- web-entity-attributes;
- web-entity-relationships coming from the web-link-relations of REST core meta-model.

Starting from this framework, a MDA initiative could further add a new level in meta-modeling approach: e.g. one could define a JEE metamodel for JEE as the platform-of-choice to implement REST services and resources.

4 RoaML Target Audience

The success or failure of any software development or technologically initiative depends on a critical "quality": the popularity that could engage a prospective massive audience.



Beta: Measuring search interest in *topics* is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific *query*, select the 'search term' option.

Fig. 6. Measuring REST vs. SOA audience

We have studied the popularity of SOA versus REST in the last years and we found the following:

- A series of statistics from Programmable Web shows a growth of the REST API from 58% in 2006 to 73% in 2011, while the SOAP API registered a decrease from 29% in 2006 to 17% in 2011 [14].
- Another study from indeed.com shows that the trend of REST jobs is increasing (1% in 2014) while the one for SOA jobs is decreasing (0.3% in 2014) [15], see figure 7.
- Google trend is showing also a growing interest in REST versus SOA based on the number of Google searches of the topic [13], see figure 6.

Judging by the success of REST we can say that RoaML has an important potential audience. The key point for the successful adoption of RoaML is represented by its simplicity and flexibility, the same principles that recommend REST over SOAP.

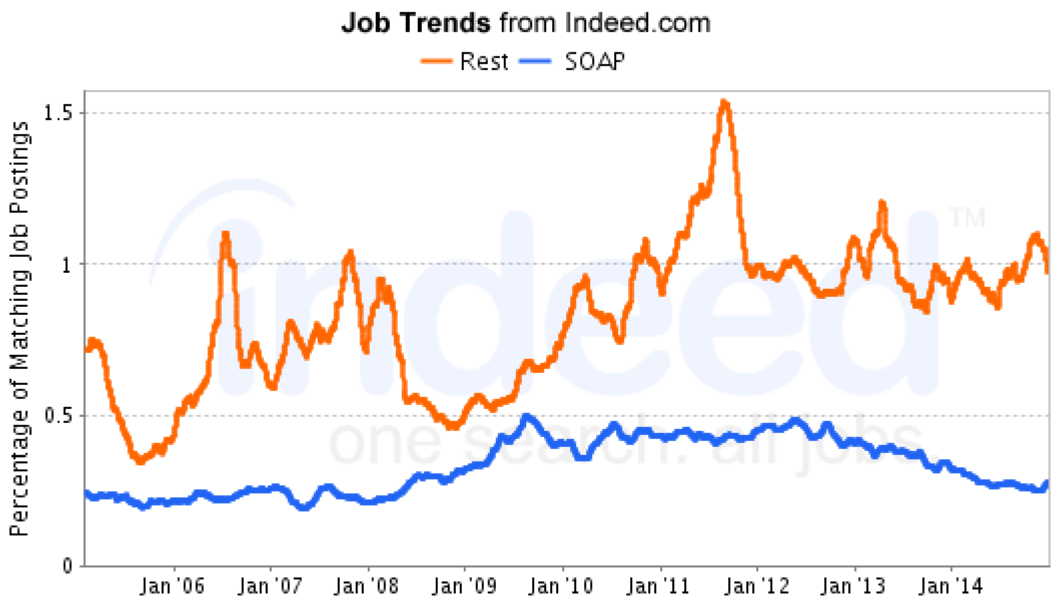


Fig. 7. Job trends from Indeed.com

5 Conclusions and Future Work

As we have argued in our paper, RoaML could be a suitable modeling language for applications based on resource oriented architecture. At this moment we are presenting just some guiding principles for a business-oriented REST/ROA metamodel, but we are planning to improve our metamodel and also to propose a MDA approach for implementing REST services and resources.

References

[1] C. Choppy, G. Reggio, A Well-Founded Approach to Service Modeling with Casl4Soa, ACM, 2010

[2] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, CHAPTER 5 Representational State Transfer (REST), http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000 Printed book

[3] T. Erl, SOA: principles of service design, Pearson Education, Inc., Boston, Massachusetts: 2008

[4] T. Erl [et.al.], Web service contract design and versioning for SOA, Pearson Education, Inc., Boston, Massachusetts: 2009, pp.25-26

[5] M. Massé, REST API Design Rulebook, Gravenstein Highway North: O'Reilly Media, Inc.m 2012, pp.15-16

[6] P. Brown, Implementing SOA: Total Architecture in Practice, Addison Wesley Professional, 2008

[7] R. Daigneau, Service design patterns: fundamental design solutions for SOAP/WSDL and restful Web services, Westford, Massachusetts: Pearson Education, Inc., 2012, pp.85-93

[8] E. Evans, Domain Driven Design: Tackling Complexity in the Heart of Software, Pearson Education, 2004.

[9] OMG, Service oriented architecture Modeling Language (SoaML) Specification Version 1.0.1, 2012

[10] K. Sookocheff, On choosing a hypermedia type for your API - HAL, JSON-LD, Collection+JSON, SIREN, Oh My!, http://sookocheff.com/posts/2014-03-11-on-choosing-a-hypermedia-format/

[11] Vinay Sahni, Best Practices for Designing a Pragmatic RESTful API, http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api , 2014

[12] S. Katoch, Design and implement RESTful web services with Rational Software Architect, http://www.ibm.com/developerworks/rational/library/design-implement-restful-web-services/ , 2011

[13]http://www.google.com/trends/explore#q =%2Fm%2F03nsxd%2C%20%2Fm%2F 0315s4&date=1%2F2007%2098m&cmpt =q&tz

[14] http://www.infoq.com/news/2011/06/Is-REST-Successful

[15]http://www.indeed.com/trendgraph/jobgraph.png?q=Rest%2C+SOAP" border="0" alt="Rest , SOAP Job Trends graph”

Cătălin STRÎMBEI has graduated the Faculty of Economics and Business Administration of Al. I. Cuza University of Iaşi in 1997. He holds a PhD diploma in Cybernetics, Statistics and Business Informatics from 2006 and he has joined the staff of the Faculty of Economics and Business Administration as teaching assistant in 1998 and as associate professor in 2013. Currently he is teaching *Object Oriented Programming*, *Multi-Tier Software Application Development* and *Database Design and Administration* within the Department of Business Information Systems, Faculty of Economics and Business Administration, Al.I.Cuza University of Iaşi. He is the author and co-author of four books and over 30 journal articles in the field of object oriented development of business applications, databases and object oriented software engineering.

Georgiana OLARU (AVRAM) has graduated the Faculty of Economics and Business Administration of Al. I. Cuza University of Iaşi in 2008. She is a PhD student in Business Information Systems with interests in modeling and design of software applications. She is a co-author in one book and author/co-author for 8 journal articles in the field of cloud computing, modelling and design of software applications and database design.