Towards an MDA Mechanism for RESTful Services Development

Christoforos Zolotas Electrical and Computer Engineering Dept. Aristotle University of Thessaloniki GR541 24, Thessaloniki, Greece Email: christopherzolotas@issel.ee.auth.gr Andreas L. Symeonidis Electrical and Computer Engineering Dept. Aristotle University of Thessaloniki GR541 24, Thessaloniki, Greece Email: asymeon@eng.auth.gr

Abstract—Automated software engineering research aspires to lead to more consistent software, faster delivery and lower production costs. Meanwhile, RESTful design is rapidly gaining momentum towards becoming the primal software engineering paradigm for the web, due to its simplicity and reusability. This paper attempts to couple the two perspectives and take the first step towards applying the MDE paradigm to RESTful service development at the PIM zone. A UML profile is introduced, which performs PIM meta-modeling of RESTful web services abiding by the third level of Richardson's maturity model. The profile embeds a slight variation of the MVC design pattern to capture the core REST qualities of a resource. The proposed profile is followed by an indicative example that demonstrates how to apply the concepts presented, in order to automate PIM production of a system according to MOF stack. Next steps include the introduction of the corresponding CIM, PSM and code production.

Index Terms—Model Driven Engineering; RESTful services; UML Profiles; Meta-modeling; Automated Software Engineering

I. INTRODUCTION

A. Core Technologies

The Representational state transfer (REST) architectural style, exhibits four principal design attributes [2]:

- 1) The Resource Oriented design, which models each concept as a resource that is addressable with a unique URI
- 2) The various types of representations a resource may have
- 3) The usage of HTTP as a web API that offers a uniform interface for all resources
- 4) The hypermedia, the "engine of application state", which is the semantic interweaving of related resources.

Once these co-exist, the produced web service falls into the third level of Richardson's maturity model [3] (RMM). Practically, these design principles allow the simple, yet efficient REST technology to gain acceptance and a bigger share of the programmable web. However, development of RESTful web services, just like any other kind of software, is prone to the common pitfalls of the established software engineering practices that may lead to project failure, late delivery, high costs or inability to serve the goals they were built for.

Aiming to reduce such risks, the Object Management Group (OMG) announced at the beginning of the previous decade a new development paradigm towards increased automation, the Model Driven Architecture (MDA) [8], which as a Model Driven Engineering instance shifts the focus of the developer from programming language code to models and transformations. The main goal of MDA is to raise the level of abstraction, keeping the developer sheltered from implementation details, thus allowing him to focus on the real problem at hand. MDA development comprises the following phases also depicted in figure 1:

- 1) Introduction of the *Computation Independent Model* (CIM), which identifies the abstract domain entities without any design or implementation details
- 2) Transformation of the CIM into the *Platform Independent Model* (PIM), which introduces the abstract design of the envisioned system, without any implementation details
- 3) Transformation of the PIM into the *Platform Specific Model* (PSM), which is an instance of the PIM enriched with concrete implementation details
- 4) Finally, production of software code from the PSM

This paper aims to take the first step towards applying the MDA paradigm to RESTful web services development in order to achieve increased automation, consistency, and faster delivery of quality code. It attempts to overcome deficiencies of other relevant approaches and introduces a UML profile serving as a meta-model for the PIM of a RESTful web service. This proposed meta-model, coupled with the related meta-models for CIM, PSM and code templates, define the essential infrastructure to accomplish automation in RESTful service development.

B. UML Profile Context

This paper describes a UML profile that is a part of the automated software engineering strategy followed within project S-CASE, an EU funded project for automating RESTful services [1]. S-CASE is a cloud based software framework aiming to facilitate fast RESTful web service prototyping and delivery by providing the core capabilities depicted in figure 1. Firstly, multi-modal input is provided to the S-CASE engine such as software functional requirements, analysis class diagrams,



Fig. 1. S-CASE main phases of RESTful services production

storyboards etc., which are then processed semantically in order to extract any involved software entities. These entities are provided as input to the MDE component, which then outputs the CIM of the service to be produced. Once the CIM is produced, the MDE component traverses through the aforementioned MDA phases, from CIM to PIM, PIM to PSM and PSM to code by means of model transformations.

Within the context of this paper the focus is on the MDE component (highlighted) and more specifically on the UML profile serving as the S-CASE engine PIM meta-model. As already mentioned (and later explained), the UML profile ensures that the semi-automatically produced system has all the core design attributes of the REST style alongside some useful design properties, such as seperation of interfaces from their implementations and uniform access to databases.

C. Paper Structure

The structure of this paper is as follows. Section 2 discusses relevant work. Section 3 presents the proposed UML profile with its constraints. Section 4 illustrates automated PIM production of a social media application. Section 5 summarizes the work performed, probes on future work and concludes the paper.

II. BACKGROUND - RELEVANT WORK

MDE has been applied to several domains and has received both appraisal (higher productivity, increased formalism and increased automation) and criticism (complexity of modeling itself, varying automation and often mostly for code generation, lack of coherence among MDE tools, etc) [7]. Thus, while one may argue that MDE cannot be considered a "panacea" for software development, the simple, unambiguous design concepts of REST, as discussed in the introductory section of this paper, predispose for succesful application of the MDA paradigm on REST applications.

Related work regarding MDE for the RESTful services domain mainly focuses on the introduction of approaches to capture RESTful services with annotated description languages. Such efforts include work by Kopecky et al. that introduce hREST [5], Maleshkova et al. that extend SAWSDL to annotate RESTful services as well [6], and Pagliarecci et al. that introduce SWSAL [10]. One should also mention work by Tavares et al. [14], who introduced a meta-model for the various semantic annotation languages of RESTful services, aiming to achieve increased interoperability among them. The above approaches mainly focus on the semantic annotation of RESTful services and their characteristics, rather than the software engineering process itself.

In a more MDE-focused research direction, Ormeno et al. [9] aspire to apply MDE to Spring Roo software artefacts in order to facilitate development of RESTful services through the IDE. Taking the problem from the applied to a more generic level, S. Shreier [12] attempted to create an initial version of a meta-model for RESTful services using as basis the *Ecore* meta-model which is based on the *EMOF* of *OMG*. There are also numerous other frameworks that attemp to model RESTful service but they either do not include hypermedia generation or they are semi-automated. Hence, this paper presents a meta-model to support third level web services in regard to RMM.

III. METAMODELING RESTFUL SERVICES

A. Grounding Definitions

Before discussing the extended UML meta-model for RESTful services, the way the *PIM* versus *PSM* on the *MOF* layer stack are conceptualized within this paper must be clarified; hence follow two definitions:

- D1: A target platform is the combination of all concrete technologies and designs used to form a system. PSM refers to such a platform.
- D2: A platform independent system is formed by the combination of technology abstractions and abstract design paradigms. PIM refers to such an abstract platform.

For example the *HTTP* verbs *POST*, *READ*, *PUT*, *DELETE* are the realization of the abstract CRUD verbs. Hence, the presented meta-model uses the concept of CRUD verbs instead.

B. The UML Profile Key Features

This subsection presents a (simplified) meta-model of the REST concepts and database schemas that frequently accompany web services. A UML profile mechanism [4] is employed that allows varying the meaning of UML meta-model by applying to it stereotypes, tags and constraints. This profile is depicted in figure 2.

The profile definition begins by modeling the resource, which is the core of a RESTful service. Abiding by the separation of concerns approach followed by the *MVC* pattern, the *Model Representation Controller* or *MRC* pattern of a resource is introduced. According to *MRC*, a resource comprises a *ResourceModel* that encapsulates resource data, some *ResourceRepresentations* each of which has a *MediaType* tag, and a *ResourceController* that has a unique URI and exposes the resource's uniform interface.

This interface is modeled by extending the UML metaclass "operation" with the stereotype *CRUDActivity*. Since there are four discrete CRUD verbs a *CRUDActivity* can be of type *CreateActivity*, *ReadActivity*, *UpdateActivity* or *DeleteActivity* to match the meaning of each CRUD verb. Every *CRUDActivity* may have some *inputMediaTypes* and/or some *outputMediaTypes*, according to its type, in order to model web communication formats such as XML or JSON.

The Hypermedia concept of REST is modeled by extending the UML meta-class "Association" with the stereotype *RelatedResource*. This extension intends to model both types of interresource relationships. That is, a resource may have related resources, or be related resource of others. The *RelatedResource*



Fig. 2. UML profile for PIM meta-modeling of RESTful services.

stereotype has two tags, the uri that links to a related resource and the ConditionSet, which is a set of conditions that must be met if the resources are to be considered related. Furthermore, to allow the association of a resource's specific CRUDActivity with one of another resource, the *RelatedCRUDActivity* data type is added to the profile. This one, apart from the ConditionSet and the uri of the related resource, also incorporates the CRUDActivity to be accessed if the ConditionSet is fulfilled. In either case, if the Conditions are met, the expected behavior of a RESTful service is to embed into the outgoing representation all the related URIs that would allow the client to forward the application state. The introduced profile models these controls with the HypermediaLink data type, which comprises an href attribute that stores the uri of accessible related resources, a rel attribute that describes the relation between the interconnected resources, and the CRUDVerb attribute that specifies which CRUDVerb must be used.

C. Business Logic and Uniform Storage

In order to allow separation of concerns, another extension of the meta-class "class" is added to the UML profile, the *CRUDActivityHandler*. This addition aims to gather the business logic of the resource in (*CRUDActivityHandlers*) and let the *CRUDActivities* handle the high level semantics of the web interface. Following the same pattern, there is one type of *CRUDActivityHandler* for each CRUD verb.

Finally, in order to introduce a uniform storage mechanism, the profile embeds the *Repository* design pattern with the *RepositoryController* stereotype. This is a dedicated component to handle the I/O operations with the underlying relational database. The relational database schema is modeled with four more extensions, the *RDBMSTable* and *Column* stereotypes, as well as the *PrimaryKey* and *ForeignKey* stereotypes to model the corresponding relational database concepts.

D. Meta-model Constraints

Typically, the presented extensions are accompanied by a list of constraints, usually in OCL. Since providing a full list of OCL constraints is out of the scope of this paper, this subsection presents a visualization of them (figure 3). These constraints depict which components of the profile may be associated with which, or composed of, other components.

The core of this meta-model, the resource, comprises a *ResourceModel* with at least one *ResourceRepresentation* and exactly one *ResourceController* that handles its web requests. Every such *ResourceModel* may have some *RelatedResources* and may be related resource of some others. The *Resource-Controller* handles the requests that must conform to the uniform *CRUD* API, so it must have one *CRUDActivity* for each *CRUD* verb of the overlying resource. Since a resource can have many representations, a *ResourceController* must have one *CRUDActivity* for every input/output media type to be supported and every media type that is to be supported must be of a *ResourceRepresentation* type that this specific *ResourceModel* is associated with. The allowed media types per *CRUDActivity* type are listed in table 1.

Moreover, since some resources may be related to non *CRUD* verb operations that are not allowed due to the uniform interface constraint, the Richardson's [11] design solution is followed. Thus a new, algorithmic, resource is added with CRUD interface in order to model the initial non CRUD operation. This case will be further clarified in the example that is presented in the next section.

Separating the business logic implementation from the uniform web API as explained earlier, each *ResourceController* has exactly one composition association with a *CRUDActivity*-*Handler* for every *CRUDActivity* it has. In turn, each *CRUDActivityHandler* accesses the unique *RepositoryController* of the



Fig. 3. Visualized constraints of the UML profile.

TABLE I I/O media types per CRUDActivity

| CRUDActivity | Input Type | Output Type |
|----------------|------------|-------------|
| CreateActivity | Yes | Yes |
| ReadActivity | No | Yes |
| UpdateActivity | Yes | Yes |
| DeleteActivity | No | No |

system in order to query the underlying database. The *RepositoryController* must have exactly one *RDBMSSelectActivity* for every *ReadActivityHandler* that is connected to it and simirarly one *RDBMSInsertActivity* for every *CreateActivityHandler*, one *RDBMSDeleteActivity* for every *DeleteActivityHandler* and one *RDBMSUpdateActivity* for every *UpdateActivityHandler*. All these handle the low level database querying. The database comprises of at most one *RDBMSTable* for every *ResourceModel* of the system. Each *RDBMStable* has one *PrimaryKey*, none or more *ForeignKeys* and as many columns as the properties of the respective *ResourceModel*.

IV. ILLUSTRATIVE AUTOMATED PIM PRODUCTION

A. The RESTMARKS Application

This section demonstrates the steps to be followed in order to automatically produce the PIM of a test application. The application at task is called RESTMARKS, a social media application that allows users to create and tag bookmarks, or search its database to retrieve bookmarks that either they own or are public bookmarks of other users. In this example it is assumed that the functional requirements (FR) of RESTMARKS are the following:

- FR1: A guest user must be able to create an account by providing a username and a password.
- FR2: A user must be able to update his/her account.
- FR3: A user must be able to add, delete, retrieve or update any bookmark of his/her account.



Fig. 4. Illustrative CIM meta-model

FR4: A user must be able to mark a personal bookmark as public or private.

FR5: A user must be able to add tags to his/her bookmarks. FR6: Any user must be able to search by tag the bookmarks.

B. From Requirements to PIM

Producing the CIM is the first step of MDA paradigm. However since this paper focuses on PIM meta-modeling and due to space limitations, only an indicative CIM meta-model is provided that is proper for the PIM meta-model that is presented.

This illustrative CIM meta-model (figure 4) comprises resources that can have related resources should a condition set is satisfied. They can be algorithmic or non-algorithmic, as already discussed, and have some properties. Moreover, the resources may have some CRUD activities, at most one of each type.

Abiding by this meta-model and taking into account the functional requirements, figure 5 demonstrates one possible *RESTMARKS* CIM. In this case there are four resources, the Account, Bookmark, Tag and TagSearch ones, whilst the TagSearch is the only non-algorithmic resource. Additionally, the Account resource has the password and a username properties(FR1), the Bookmark resource has a URL and scope (FR3-FR4) and the Tag one has a textual description property (FR5). Moreover, the assigned CRUD activities to the Account resource are the update and create one (FR1-FR2) whilst to the Bookmark and Tag ones, all the possible CRUD



Fig. 5. Illustrative RESTMARKS application CIM

activities. Finally, figure 5 demonstrates the relations among the resources.

C. Identifying the Resources

The process begins with transforming the CIM resources to their PIM counterparts, which comprises one *ResourceModel* one *ResourceController* and some *ResourceRepresentations*. In RESTMARKS case the identified resources are the *Account*, *Bookmark*, *Tag* and *TagSearch*. All these components are interconnected as prescribed by the meta-model constraints, so every *ResourceController* has a unidirectional association with the corresponding *ResourceModel*, which in turn has a unidirectional association with its *ResourceRepresentations*.

Once the resources are added to the PIM, their properties are added to the respective*ResourceModels*. In this case the *AccountModel* has the *password* and *username* properties, the *BookmarkModel* has the *url* and *scope* and the *TagModel* has the *description*.

D. Identifying the Relations Among Resources

The next step is to apply the identified relations among the resources. In RESTMARK's CIM case, the *AccountModel* is not a related resource of any other resource, but has as related resource the *BookmarkModel*. Therefore, a *RelatedResource* unidirectional association is added from *AccountModel* towards *BookmarkModel*. In the same pattern, figure 6 demonstrates the semantic interweaving of all the resources.

E. Adding CRUD Verb Activities

The next step is to add every *CRUDActivity* that is allowed on a specific CIM resource to the respective PIM *Resource-Controller*. Hence, the *CreateAccount* and *UpdateAccount CRUDActivities* are added to *AccountController* and likewise the *createBookmark*, *readBookmark*, *updateBookmark* and *deleteBookmark* CRUDActivities to *BookmarkController* and so on. Moreover, following the meta-model constraints, each *ResourceController* must have exactly one unidirectional composition association with a *CRUDActivityHandler* for every *CRUDActivity* of the same type. Thus, *AccountController* has a *createAccountActivityHandler* and an *updateAccountActivity*-*Handler*, while the *BookmarkController* and the *TagController* have every type of *CRUDActivityHandlers*.

F. Creating the Relational Schema

Creating the underlying database is the next step, once the *ResourceModels* are in place. Thus, for every non algorithmic *ResourceModel* of the *PIM*, an *RDBMSTable* is added to it. Therefore, the tables *Account*, *Bookmark* and *Tag* are added to the *RESTMARKS PIM*. Subsequently, for each *ResourceModel* property, a column is added to the respective table. Therefore, the columns *username* and *password* are added to the *Account* table, the columns *url* and *scope* to table *Bookmark* so on.

However, for relational reference, each *RDBMSTable* must have exactly one primary key. Therefore, a new column as a primary identifier is added to each table with the same name as the table, post-fixed with *id*. Hence the primary key *AccountId* is added to table *Account*, and *BookmarkId* and *TagId* to tables *Bookmark* and *Tag* repsectively. Finally, in order to achieve consistent mapping among the schema tables and the *ResourceModels*, one more property is added to each of them with the same name and type as the primary key of each schema table. Therefore, the property *AccountId* is added to *AccountModel*, *BookmarkId* to *BookmarkModel* and *TagId* to *TagModel*.

Establishing proper cross-reference among the interweaved resources with foreing keys, is the last step to produce the database schema. For every resource X that is a related resource of another one Y, the primary key of that table Y is added as foreign key to the table X. In this case, the foreign key *BookmarkId* is added to table *Tag* and the foreign key *AccountId* to table *Bookmark*.

G. Creating the RepositoryController

The final step towards PIM production is to create the schema controller so that all *CRUDActivityHandlers* have access to the persistent storage. Thus, the *RDBMSController* is added to the system. As prescribed in the profile constraints, the *RepositoryController* must have one *RDBMSSelectActivity* for every read activity, one *RDBMSDeleteActivity* for every delete activity and one *RDBMSUpdateActivity* for every update activity. In this context the *createAccount RDBMSUpdateActivity* as well. In a similar manner, the respective *RDBMSSelectActivities* are defined for the other resources.

Once all the above steps are executed, by an algorithm that conforms to the presented meta-model, the RESTMARKS PIM is completed (figure 6). This PIM would then be the input to another PSM generation algorithm that conforms to an appropriate for this PIM, PSM. That PSM in turn, would contain the needed meta-data for code production based on code templates. In general, should the MDA paradigm is followed by using the presented profile, once the developer completes creating the CIM, the rest is automated up to the PIM creation. Should the PSM and code generation components are defined likewise, the PSM and code generation would be semi-automatically produced as well. It must be noted though that depending on the case, the completeness of the produced system may greatly vary and the developer will



Fig. 6. RESTMARKS application produced PIM.

have to fill in the details that where not described sufficiently by the UML profile such as authentication.

V. CONCLUSION AND FURTHER WORK

This paper introduces a RESTful service PIM meta-model that aids the semi-automated development of services that respect REST design qualities, such as resource oriented design with uniform CRUD interface and embedded hypermedia links. It demonstrates as well a set of steps to be taken in order to produce the PIM of an indicative application called RESTMARKS.However, it is only one of the steps towards applying MDA in RESTful development. Next steps include the definition of a full MDA mechanism that also includes the CIM, PSM and code generation steps as well.

Acknowledgments.: Parts of this work have been supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission.

REFERENCES

- [1] FP7 Collaborative Project S-CASE: Scaffolding Scalable Software Services. http://www.scasefp7.eu.
- [2] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University Of California, 2000.
- [3] M. Fowler. Richardson Maturity Model: steps towards the glory of REST. http://martinfowler.com/articles/richardsonMaturity Model.html, March 2010.

- [4] L. Fuentes-Fernadez and A. Vallecilo-Moreno. An introduction to uml profiles. *The European Journal for the Informatics Professional*, 2:6–13, 2004.
- [5] J. Kopecky, K. Gomadam, and T. Vitvar. hrests: An html microformat for describing restful web services. In WI-IAT '08, volume 1, pages 619–625, Dec 2008.
- [6] M. Maleshkova, J. Kopecky, and C. Pedrinaci. Adapting sawsdl for semantic annotations of restful services. In On the Move to Meaningful Internet Systems: OTM 2009 Workshops, pages 917–926, 2009.
- [7] P. Mohaghehi and V. Dehlen. Where is the proof? a review of experiences from applying mde in industry. In *Model Driven Architecture -Foundations and Applications*, volume 5095, pages 432–443, 2008.
- [8] OMG, http://www.omg.org/mda/. Model Driven Architecture, 2000.
- [9] E. Ormeno, M. Lund, L. Aballay, and S. Aciar. An uml profile for modeling restful servicecs. In ASSE 2012, pages 119–133, 2012.
- [10] F. Pagliarecci, L. Spalazzi, and G. Taccari. Application of swsal in semantic annotation of restful web services. In *INVIT2012*, pages 11– 18, 2012.
- [11] L. Richardson and S. Ruby. RESTful Web Srvices. O'Reilly, 2007.
- [12] S. Sshreier. Modeling restful applications. In WS-REST 2011, pages 15–21, 2011.
- [13] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Longman, 2009.
- [14] N. A. C. Tavares and S. Vale. A model driven approach for the development of semantic restful web services. In *IIWAS 2013*, page 290, 2013.