Towards an Interpretation Framework for Assessing Interface Uniformity in REST

Michael Athanasopoulos
National Technical University of Athens, Greece
athanm@softlab.ntua.gr

Kostas Kontogiannis
National Technical University of Athens, Greece
kkontog@softlab.ntua.gr

Chris Brealey
IBM Toronto Lab
Ontario, Canada
cbrealey@ca.ibm.com

ABSTRACT

Interface uniformity is regarded as one of the most distinctive features of the REST architectural style among other network-based styles, because of the specific set of restrictions it imposes on the behavior paradigms of interacting components. However, in practice conforming to the REST's uniform interface constraint in Web-based services most often proves to be a difficult task, as identified by a number of researchers and practitioners. This implementation and conformance difficulty can be partly attributed to the lack of a systematic conceptual framework that could be used to interpret abstract architectural restrictions of interface uniformity to practical design decisions and strategies being generalized as interface design criteria. These criteria could be then mapped to domain-specific techniques that provide the context for guiding and/or examining the level of uniformity of a REST-based API. In this paper, we discuss such a conceptual framework and a collection of criteria that can be used to assess in a practical way as to whether a specific REST-based API conforms to the uniform interface constraint. As a proof of concept, we evaluated the proposed framework and its associated methodology by applying it to a collection of indicative public Web service APIs.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architecture - *domain-specific architectures*.

General Terms

Design, Theory, Measurement

Keywords

REST, Uniform Interface, Web Services, API.

1. INTRODUCTION

Interface-level generality of service systems allows for abstraction and uniformity of interaction paradigms and independence from implementation technologies and even from application and domain-specific point of view. More specifically, the concept of uniformity promotes visibility, which in the specific context of REST interface definition can facilitate a significant reduction in the cost of integration of components. Furthermore, abstraction facilitates higher degrees of decoupling and independence between components, leading to architectures that demonstrate significantly improved levels of evolution-tolerance and reusability. According to [1] the uniform interface constraint advocated by REST is considered to be as "*the central feature that distinguishes REST from other network-based architectural styles*". However, this constraint proves to be one of the most difficult principles to evaluate, and is probably one of the most debated concepts in the REST community.

According to REST's definition, the Uniform Interface constraint encloses four architectural sub-constraints that should encompass the behavior of components at interface level, namely: a) identification of resources, b) manipulation of resources through representations, c) self-descriptive messages and, d) hypermedia as the engine of application state. Fielding's dissertation provides a discussion related to the above-mentioned sub-constraints without explicit definitions. Consequently, over the years, these constraints have been subject to multiple interpretations on how they should be realized in a RESTful architecture. In this paper, we introduce and discuss a goal-oriented conceptual framework that consists of three layers that pertain to architecture, design and instantiation context. At the architecture layer the framework complies with constraint properties presented in [1] while at the design and instantiation layers presents a collection of guiding features that can be used to evaluate and assess as to whether a REST-based API truly conforms to the uniform interface constraint for a given application-domain and deployment scope. In this respect, the proposed framework aims to take into account not only the features that need to be evaluated but also the context and the domain in which the REST-based API is used. In this work we particularly focus and discuss the second layer that introduces an initial collection of such compliance evaluation design criteria. The proposed framework could also allow for the development of shared understanding between interacting stakeholders during the construction and maintenance of RESTful systems. Furthermore, such a conceptual framework could be applied to other REST's constraints in order to provide a disciplined method for examining systems with regards to the anticipated properties of a RESTful architecture, given the level of its conformance to each of the constraints.

This paper is organized as follows. In the Section 2, we outline related publications on the subject of REST's uniform interface constraint. In Section 3 we discuss the proposed framework in the form of a goal-tree model and present the set of interface design criteria per sub-constraint. Section 4 presents an evaluation of the framework by applying to a set of indicative public APIs. Finally, Section 5 concludes the discussion and provides pointers for future research.

2. BACKGROUND AND RELATED WORK

According to [1] the *Uniform Interface* constraint, along with the *Layered System* and the *Code-On-Demand* constraints constitute the subset of REST's constraints that were added to the early Web architecture to guide its evolution. Fielding also briefly discusses the trade-off that exists when interface definitions need to be matched to generic interface requirements and recognizes that this could result in a degradation of efficiency with regards to application-specific needs which could be better served with the definition of arbitrary interfaces. On the other hand, the standardization of interfaces decouples implementation from interaction, adding a level of abstraction or "insulation" that preserves the stable behavior of communicating components which becomes a matter of vital importance when large-scale integration is required. Furthermore, the demand of uniform interfaces between components in a RESTful architecture is also closely related to the concept of a resource as (re)defined in the context of REST. Centered on the idea of the resource as the conceptual target of a hypermedia reference and away from body-format or content-type definitions, the uniform interface essentially standardizes the model of interaction with the underlying semantics of a resource without the need of understanding these particular resource-specific semantics. In this respect, capabilities (such as data and functionality) can be uniformly addressed and manipulated, while the prior knowledge needed for such interactions can be minimized into commonly accepted and well-defined artifacts (communcation protocols, metadata, message formats, etc). However, in practice and especially in the context of Web-based information services, it as a common phenomenon for architects to (intentionally or not) fail to abide by the uniformity that REST architectural style requires, although they still denote their service systems as RESTful. This fact has been identified by a number of REST researchers and practitioners and it has also led Fielding to post several rules for designing REST APIs on a blog entry [2] most of which, directly or indirectly, reflect restrictions imposed by the uniform interface constraint. These rules have been examined and further analyzed in related discussions in the REST community and have been used as input for our work.

Another interesting approach in understanding the extent that Web-based systems conform to REST, is a maturity-based model usually referred to as the *Richardson Maturity Model (RMM)* [3]. RMM includes four levels of maturity which relate significantly to the level of conformance to the uniform interface constraint. Similarly but explicitly associated with uniform interface's sub-constraints [4] presents a classification of HTTP-based APIs. The classification describes five types of interfaces, namely *WS-**, *RPC URI-Tunneling*, *HTTP-based Type I*, *HTTP-based Type II*, and *REST*. The API types are also examined and characterized as to the effects they may have on properties induced by the level of conformance to the sub-constraints of the uniform interface.

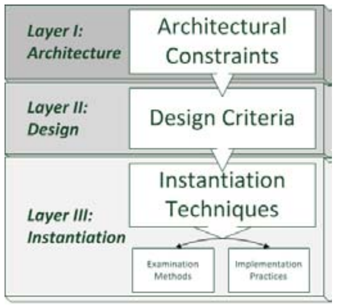

Figure 1. High-level meta-model of the Uniform Interface Conceptual Framework.

3. UNIFORM INTERFACE CONCEPTUAL FRAMEWORK

3.1 Meta-model Description

The *Uniform Interface Conceptual Framework (UICF)* discussed in this paper follows the structure of a simplified goal tree [6] that can be divided into three conceptually distinct layers: *Architecture*, *Design* and *Instantiation*. UICF extends the semantics of nodes to capture layer-specific types. Figure 1 presents a high-level meta-model of UICF. Layer I includes *Architectural Constraint* nodes, layer II includes *Design Criterion* nodes and layer III contains *Instantiation Technique* nodes that represent either examination methods or implementation practices given specific ecosystem context and analysis scope.

Architectural constraints in the first layer have direct reference to REST's definition according to [1] while design criteria in the second layer constitute practical interpretations of these architectural constraints but in a technology-agnostic way. These criteria often represent compromises or conventions after meticulous argumentation over how to implement abstract architectural concepts in order to obtain a uniform interface without becoming context or technology-specific. Specifically, in the current version of UICF, we populated the design layer with interpretations extracted through reviewing the literature and publications on REST and resulted to a set of criteria that cover a significant spectrum of issues that a REST designer faces. The major differentiating feature of design criteria at the second layer of the proposed framework, when compared to architectural constraints of the first layer (see Figure 1) is that it lowers the level of abstraction by introducing a set of identifiable, concrete practice-oriented conceptual units in order to guide or assess design in a way that is technology-agnostic while not being technology-ignorant. Finally, the design criteria of the second layer are manifested as instantiation techniques in the third layer. In this respect, we can consider that the instantiation layer is populated with realization-level configurable techniques which can be used to either examine the conformance of an interface to the REST architectural style, or guide the implementation of systems in order to conform to it.

3.2 Framework Feature Model

The proposed framework (UICF) is depicted in Figure 2. This section focuses on the design criteria defined by UICF in order to assess interface uniformity in REST.

3.2.1 Identification of resources

In the context of this work, the identification of resources sub-constraint is examined in two aspects: a) with respect to interface-level design of the resource model, and b) with respect to the assignment of identifiers.

ID1: Resources with distinguishable informational content. REST interface design and resource modeling should be based on content-oriented conceptualizations of important resources, instead of information-poor action dereferences. Addressable resources should generally capture explicit and distinguishable informational content and may provide references to other resources. In this sense, there should be a consistent underlying mechanism to map the informational content exchanged between components to application-specific domain entities, processes, non-functional characteristics, or combinations of them.

ID2: Disciplined functionality exposure. Resources should have state which is transferred through representations, conveying semantically interesting information. The manipulation options of the resource state in conjunction with the semantics of the representation elements should be the preferable way of exposing functionality. Such a mapping often requires a substantial level of isomorphism between interface modeling and implementation. However, there are also cases where functionality exposure may be addressed by identifying resources that reflect procedural or execution instance semantics.

ID3: Universal naming mechanism. The identification mechanism used to assign identifiers to resources should be consistent and universal across the ecosystem in which the designed system is or will be deployed. Such an identification mechanism would provide a generic meta-model for particular identifier structures that will be actually used by the system. Furthermore, the naming mechanism should allow for the preservation of identification and interaction as two separate concerns. Finally, universality also indicates that every stakeholder in the ecosystem is assumed to have or to be able to acquire full understanding of the mechanism.

ID4: Identification transparency. Identifiers should be substitutable at any point in time without any side-effects or impact expected on the client-side operation, provided that the set of entry point identifiers as well as the set of possible access mechanisms are stable. Identifiers should not be overloaded with any other type of resource-specific information (e.g. action semantics over resources, resource type semantics, resource relationships etc.) other than the minimal set of data required for the identifier's resolution (i.e. indication of possible mechanisms to dereference it). Also, no other information apart from the universal naming mechanism should be available on how to construct and populate patterns or templates of resource identifiers before run-time.

3.2.2 Manipulation of resources through representations

The representations of the resources, the manipulation through such representations and the relationship between manipulation and representation semantics are addressed in the proposed framework by the following criteria.

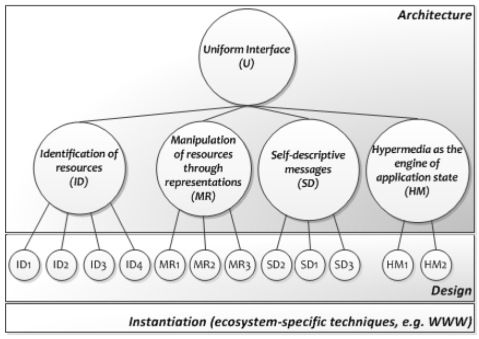
Architecture
Uniform Interface (U)
Identification of resources (ID)
Manipulation of resources through representations (MR)
Self-descriptive messages (SD)
Hypermedia as the engine of application state (HM)
ID1
ID2
ID3
ID4
MR1
MR2
MR3
SD2
SD1
SD3
HM1
HM2
Design
Instantiation (ecosystem-specific techniques, e.g. WWW)

Figure 2. Uniform Interface Conceptual Framework.

MR1: Identifiable, universal and expressive content types. The types of the data exchanged through resource representations should conform to universal, well-defined schemas, with definitions accessible to anyone in the ecosystem. Content type descriptions should guide the interacting parts on both the syntactic and the semantic level as to how to parse, understand and process representations for the objectives of the interaction. These descriptions may evolve over time and each family of versions should be identified under a unique identifier.

MR2: Universal resource-related actions. The possible intentions conveyed through retrieval and manipulation requests should be normalized into a set of actions that have universal semantics across all identified resources. These semantics should be consistently mapped to communication protocols used, respecting the expected properties of protocol-defined actions. Furthermore, all the recipients of the requests (that is, intermediate components as well as the end-server) should be able to understand these universal actions and act according to their roles.

MR3: Manipulation consistency. Protocols have defined semantics and properties, usually being broad enough to cover a convenient spectrum of usage. However, interface descriptions and content types often associate parts of their specifications with particular protocol(s) and may dictate specific protocol usage patterns, assigning in this way ad-hoc semantics to such interactions. These ad-hoc semantics may violate protocol-defined semantics or expected properties (e.g. idempotence). To avoid such confusion and preserve consistency, interface descriptions and content types being used, should be fully compliant with the protocol definition and may only fix the details or disambiguate parts of protocol semantics that may be unclear when used under special conditions, by establishing non-conflicting conventions across the system stakeholders.

3.2.3 Self-descriptive messages

Self-descriptiveness of messages is evaluated utilizing the semantically distinct descriptive annotation units that can be enclosed in the exchanged messages and their interpretation.

SD1: Protocol-based interaction control. The semantics of the intent of interactions (requests and responses) can be conveyed through protocol-defined control data (e.g. action, responses, caching etc). In this way, recipients' behavior may be directed as described in the communication protocol. When a mapping of application-logic intents to the protocol's control semantics is not possible or too difficult to achieve, protocol selection could be re-examined. In this respect, no custom, application-specific control data should be used to guide the control of interaction.

SD2: Universal resource and representation metadata. Metadata provide a generic way of conveying resource and representation descriptions. All the message recipients should be able to recognize, distinguish and understand this metadata, given the context of the architecture. Usually such metadata descriptions are included in the definition of the communication protocols. The descriptions may also refer to external binders (e.g. IANA's repository of media types) or provide extension points so that orthogonal, well-defined description formalisms can be employed.

SD3: Consistent interpretation of the descriptive information. The interpretation of any descriptive annotation units (e.g. control data, metadata) included in the exchanged messages should be consistent with the semantics that are prescribed by their specifications (e.g. the interpretation of representation data should be according to the corresponding published media type specification).

3.2.4 Hypermedia as the engine of application state

Hypermedia should be the exclusive way of advancing application state when interaction is required. Two fundamental criteria are described to indicate the conformance to the constraint.

HM1: Hypermedia-enabled messages. Data and metadata exchanged through messages should implicitly or explicitly manifest the existence of hypermedia references. These references can be then used to reach consequent application states. In this respect, there should be no other way for resource identifiers to be constructed other than by utilizing the exchanged information - with the exclusion of entry point resources. However, not all messages have to be hypermedia-enabled if this does not affect the determination of the consequent application state.

HM2: Hypermedia controls expressiveness. Hypermedia controls that are present in the exchanged messages should be adequately expressive in order to guide the transitions of application state and fully address domain-level concerns. Such hypermedia controls may take the form of a) embedded elements in the messages with semantics defined by the media type's specification, b) references to additional hypermedia markup defined by orthogonal specifications, or, c) products as results of computation over dynamically delivered data including ad-hoc processing instructions.

4. EVALUATION

We applied the proposed UICF framework on three popular "REST APIs", namely *Flickr*, *Twitter* and *Sun Cloud* APIs which are, in our opinion, indicative of how diverse is the set of APIs labeled as "REST". Examination methods were constructed for each criterion as instantiation techniques in the context of WWW and the results of the examination are presented on Table 1.

Flickr API does not meet most of the set criteria leading to the violation of all the uniform interface constraints. Twitter API performs better but fails to fully conform to any of the sub-constraints, too. Sun Cloud API does not meet the second self-descriptiveness criterion (SD2), since the scope of the evaluation was the WWW and the media types used by the API are not registered to IANA's repository of media-types. However, it should be noted that REST does not require the formal standardization of the media types used, as commented in by Fielding [5]. In this sense, MR1 holds for Sun Cloud API: the content types used are identifiable, expressive and have ecosystem-wide available descriptions, although not registered to IANA's repository. However, it must be noted that conformance to the uniform interface constraint should not be regarded as an evaluator of the overall quality of a system, as the constraint may be intentionally violated as a design decision for meeting specific system requirements.

Table 1. Flickr, Twitter and Sun Cloud APIs examination.

API	**ID**				**MR**			**SD**			**HM**	
	ID1	**ID2**	**ID3**	**ID4**	**MR1**	**MR2**	**MR3**	**SD1**	**SD2**	**SD3**	**HM1**	**HM2**
Flickr	-	-	+	-	-	+	+	-	+	-	-	-
Twitter	+	-	+	-	-	-	+	+	+	-	-	-
Sun Cloud	+	+	+	+	+	+	+	+	-	+	+	+

5. CONCLUSIONS AND DISCUSSION

Interface uniformity of interacting components is a crucial property that should be imposed over components in order to construct a RESTful architecture. However, uniform interface constraints can be quite difficult to be translated into specific design practices or strategies, and they have been subject to various and sometimes conflicting interpretations. In this paper, we presented an initial version of a conceptual framework for assessing and evaluating interface uniformity, in the form of an extensible three-layer goal tree, called Uniform Interface Conceptual Framework (UICF). The layered approach that UICF follows allows for the separation of concerns (architecture, design, and instantiation) and aims to promote the shared understanding between stakeholders involved in the development of RESTful or generally "Web-inspired" architectures. Furthermore, UICF can be used as a systematic approach for the examination of such systems with regards to the level of conformance to the uniform interface constraint, given their context. In this context future work includes the design and development of an automated tool that could assess the level of conformance of a REST API with respect to the uniform interface constraint as discussed in REST.

6. REFERENCES

[1] Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[2] "REST APIs must be hypermedia-driven", http://roy.gbivn.com/untangled/2008/rest-apis-must-be-hypertext-driven

[3] "Richardson Maturity Model", http://www.crummy.com/writing/speaking/2008-QCon/act3.html

[4] "Classification of HTTP-based APIs", http://www.nordsc.com/ext/classification_of_http_based_apis.html

[5] Fielding, R. T. Comments on self-descriptiveness of media types, http://tech.groups.yahoo.com/group/rest-discuss/message/6594

[6] Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R. "Reasoning with Goal Models". In Proc. of 21st Int. Conf. on Conceptual Modeling, Tampere, Finland, October 7-11, 2002, pp.167-18