A Maturity Model for Semantic RESTful Web APIs

Ivan Salvadori
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, SC - Brazil
ivan.salvadori@posgrad.ufsc.br

Frank Siqueira
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, SC - Brazil
frank@inf.ufsc.br

***Abstract*—Web APIs provide interfaces for interaction among systems based on the existing infrastructure for hosting Web sites and applications. The REST architectural style is the most employed approach for building Web APIs. However, the flexibility provided by REST may result in implementations with low quality design, limited reuse and poor documentation. This paper describes a maturity model for classifying Web APIs, aimed at promoting the adherence to REST architectural principles and the adoption of semantic Web technology in order to improve the design, reuse and documentation of Web APIs.**

***Keywords*-Maturity Model; REST; Web API; Linked Data; Semantic Web;**

I. Introduction

REST is a very popular architectural style for integration of Web-based applications, which allows sharing and reusing information through distributed systems. Large-scale applications based on the REST principle must be implemented using the appropriate strategies and mechanisms in order to produce systems that are easy to develop, reuse and maintain. The integration interfaces provided by these systems, called Web APIs, have an important influence on the characteristics of the resulting implementation.

According to Sjoberg, Dyba and Jorgensen [1], the vast majority of research studies are aimed at technology evolution, while technology evaluation is less investigated. Maturity models provide a useful reference for evaluation, providing criteria for classifying existing implementations and the employed technologies. Heitmann et al. [2] present an empirical study that evaluates the level of adoption of semantic technology for application development and identifies the main technological challenges for its adoption. One of the main challenges identified by the authors of this study is the lack of standards and guidelines for developing applications with semantic capabilities. Therefore, a maturity model able to evaluate semantic applications can provide important guidelines for application developers. In addition, Tahir and Macdonell [3] conclude that most of the existing quality metrics are targeted at general-purpose object-oriented systems. Other categories of systems, including Web-based systems, are not sufficiently explored. The existing maturity models for Web API design found in the literature [4][5], though, do not take into account all characteristics required for developing implementations with semantic support.

This paper introduces a new maturity model for RESTful Web APIs that evaluates three complementary and orthogonal dimensions: Design, Profile and Semantics. These dimensions represent fundamental characteristics for an efficient and reliable integration among systems. This work also identifies strategies for achieving the highest maturity levels of the proposed model.

The remainder of this paper is organized as follows. Section II presents the main concepts required for understanding the proposed model. The related work is described in section III. Section IV presents a new maturity model for Web APIs. Some support technologies for Web API development are evaluated in section V. Finally, section VI summarizes the outcomes of this work and points out some open areas for further development.

II. Background

This section describes the main concepts related to the maturity model for Web APIs proposed in this paper.

A. Semantic Web

The Semantic Web [6] is an extension of the current World Wide Web, in which data available on the Web is semantically described. The Semantic Web allows not only humans, but also machines to infer the meaning of data published on the Web, and also facilitates data integration and reuse. It is a natural evolution from a Web based on hypertext, targeted at human beings, toward a *Web of data*, which can be interpreted by machines due to the association of published content with their semantic description. The combination of Web content with semantic descriptions creates an interconnected information structure known as Linked Data [7].

Data available on the Semantic Web must be identified by HTTP Uniform Resource Identifiers (URIs) and must be associated with semantic descriptions built using standard technologies recommended by the World Wide Web Consortium (W3C). One of such standards is the Resource Description Framework (RDF) [8], which is a method for information modeling that can be employed for describing resource semantics. By using RDF, information is structured as triples *subject-predicate-object*. Subject represents the resource that is being described, predicate is a characteristic of the subject, and object is a value assigned to the characteristic of the subject. The available data must also have hyperlinks to related content, in order to enable the navigation through the Web of data.

B. REST

REpresentational State Transfer (REST) is an architectural style that defines a set of constraints aimed at improving the performance, availability and scalability of Web-based distributed systems [9]. It is based on the traditional client-server paradigm, in which requests issued by clients are executed by a server.

A RESTful implementation is one that follows the constraints defined by the REST architectural style. REST defines a uniform interface for system components based on four constraints: resource

978-1-4673-7272-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICWS.2015.98

identification, handling resources through representations, self-described messages and Hypermedia as the Engine of Application State (HATEOAS).

Resources are abstractions that represent information handled by RESTful applications. A resource must be identifiable and accessible through a generic interface. Resources are not directly accessed; instead, they are seen through a representation, which is a snapshot of the state of a resource at a given time. Representations of a given resource may be available in different formats, such as XML, JSON, HTML, and so on. Representations are obtained by issuing requests to Web services provided by a RESTful application.

RESTful services must have stateless behavior, i.e., the server does not keep information regarding the requests issued by clients. This constraint is vital to improve the scalability of the system. Stateless behavior requires self-described messages, i.e., the request must contain all the required information for being processed by the server.

The HATEOAS constraint defines that client interaction must be guided through hypermedia controls, which provide mechanisms for querying, storing and handling information on resources. The HATEOAS principle implies that resource representations must provide not only resource information, but also hypermedia controls to guide client interaction, informing the operations that may be performed on the resource and providing links to other resources which are made available by the server.

C. Web API Design

Web APIs consist in programmatic interfaces that provide access to Web Services for remote clients, allowing the construction of distributed applications based on the Web infrastructure [10]. The design of a Web API has a direct influence on the quality of communication between a server and its clients, given that it defines how information is modeled, transmitted and interpreted.

The most adopted design styles for building Web APIs are based on Remote Procedure Calls (RPC) and on Resources. Web APIs designed based on RPC style employ the HTTP protocol to request operations, which are processed by a Web Service addressed by a single URI. The service receives a data payload and executes the requested operation using a set of provided parameters. The RPC style does not follow the semantics of the HTTP protocol; services are requested through a HTTP POST method and the result of execution is contained in a payload sent to the client.

The design style based on Resources creates an interrelated set of data, which becomes available through the Web API. Each resource is addressed by a URI, which allows the execution of operations on the resource. Operations are often limited to CRUD (Create, Read, Update and Delete) procedures applied on resource data.

The best design strategies for modeling a Web API make use of the original semantics of the data transfer protocol (e.g., HTTP). This paper identifies the existing strategies for modeling a resource-based Web API and classifies them according to the level of quality of the obtained implementation, resulting in a maturity model to guide the construction and to evaluate the quality of Web APIs.

D. Semantic RESTful Web APIs

The Semantic Web is based on principles and technologies that may also be applied for the construction of semantically enriched Web APIs. A Semantic Web API is one in which data and operations are semantically described. APIs based on RESTful resources are able to profit from semantic technology, given that there is no widely accepted standard for describing data and operations provided by RESTful services.

One enabling technology for implementing Semantic RESTful Web APIs is JSON-LD [11], which is a W3C standard that extends the JSON format, allowing the addition of Linked Data to resource representations. Resources described in JSON-LD are also valid JSON documents, allowing the reuse of the existing libraries and tools for processing JSON data. One of the main goals of JSON-LD is to allow the construction of Web APIs that provide access to resources that are semantically described by controlled vocabularies associated to a given application domain. The description of resources through a controlled vocabulary establishes a clearly defined context related to the information exchanged with client applications, increasing their understanding of data semantics.

Although JSON-LD represents a step forward toward the addition of hypermedia controls to resource representations, it does not provide means to describe the available operations to handle resources [10]. An alternative that allows wider use of hypermedia control is provided by Hydra [12], which extends JSON-LD with a vocabulary aimed at describing operation semantics. Hydra allows the exchange of information in JSON, using the semantic context provided by JSON-LD with the addition of instructions to describe the operations supported by a resource. Besides enriching the information made available through the Web API, Hydra offers mechanisms for documenting the whole set of resources accessible through the Web API.

E. Web API Description and Documentation

Interaction with Web APIs may require the execution of complex communication steps between a client and the server. The communication flow is often composed by a sequence of requests that must follow a pre-established order in the aim of executing a complex activity in a given application domain. A Domain Application Protocol (DAP) allows the description of interactions among entities in a distributed system, guiding the execution of a business process [13].

The HTTP protocol provides a uniform interface for handling Web resources among different platforms, standardizing the semantics of protocol messages. However, HTTP semantics is not sufficient for describing all details of a given application domain [13]. One solution for this problem is the adoption of Profiles [10] to describe application and protocol semantics. A profile describes a Web API in a format that can be understood by humans and by software agents, allowing the construction of intelligent and autonomous clients.

Technologies such as JSON-LD and Hydra can be employed for description of Profiles [10]. The documentation format for Web APIs offered by Hydra allows the description of resources with their corresponding properties and operations. Complex interactions, though, cannot be described in a profile, requiring the use of hypermedia controls to guide the client.

F. Quality Criteria

Meskens [14] presents a methodology to establish a diagnosis based on quality criteria, which helps developers to identify if an application should be reengineered or completely redeveloped. Flexibility, maintainability and testability are considered the main quality factors, which result in many criteria such as: consistency, self-documentation, modularity, simplicity and conciseness. Metrics such as coupling, size and cohesiveness, are also taken into account by the methodology. Four levels of abstraction are identified by the methodology. The implementation level identifies components (e.g., files, modules, variables) and explains how they are tied together, abstracting the programming language chosen for system development. The structural level reveals how parts of the system will influence the other parts. The functional level helps to explain the logical relationships among components. At last, the domain level provides an interpretation of the functional behavior of components with concepts specific to the application domain.

According to Daud and Kadir [15] and Chahal and Singh[16], the most employed metrics for evaluating the quality of computer systems are cohesion, coupling and complexity. Coupling measures the dependency among system modules. Cohesion evaluates if the internal features of a module are closely related. Complexity derives from coupling and cohesion, and can be defined as the difficulty to understand and manage a system [17]. These metrics are employed by Tahir and Macdonell [3] and by Daud and Kadir [15]. The first two investigate the characteristics related to dynamic metrics, i.e., metrics applied during system execution [18]. The authors state that complexity and maintainability are the most explored aspects by dynamic metrics, while aspects such as testability and reusability are not given much attention. Daud and Kadir [15], on the other hand, classify applications based on SOA (Service Oriented Architecture) principles. These authors state that modularity and low coupling have an important role in SOA applications, given that these applications establish service contracts and behavioral agreements. These authors divided the development process in four phases: requirements, design, implementation and maintenance. However, the majority of metrics established for SOA systems focuses on the implementation and design phases.

III. Related Work

Maturity models are useful to evaluate the quality of software design and the resulting implementation. This section describes two proposals found in the literature that present maturity models for evaluation of Web APIs.

A. Richardson Maturity Model

Richardson Maturity Model (RMM) [5] classifies Web APIs in four maturity levels according to their design, as shown by Fig. 1(a). The lowest level consist in using HTTP just as a data transfer protocol, without following the semantics of HTTP methods and REST architectural principles. The most common format for data representation is XML, with a Plain Old XML style (POX). Interactions are done in RPC style, providing access to all services through a single access point.

The second level comprises Web APIs organized following the concept of Web resources. Information accessible through the API is modeled as multiple resources. Every single resource is uniquely identified and addressed, allowing the client to handle resources individually. Each resource can be accessed through its own set of operations.

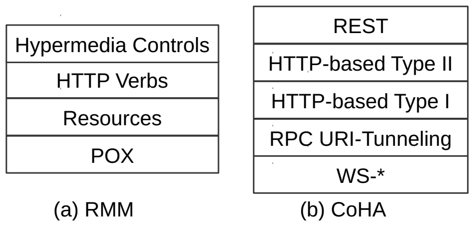

Figure 1. (a) Richardson and (b) CoHA maturity models

The third maturity level enforces the conformance with the semantics of the HTTP protocol. It requires the correct use of HTTP messages to handle resources accordingly, i.e. POST creates a resource, PUT modifies it, GET retrieves it and DELETE removes the corresponding resource. HTTP status codes must also be used in response messages to describe with consistency the result of operations performed on resources.

The highest maturity level requires adherence to the HATEOAS principle through the provision of hypermedia controls by the Web API. The provision of hypermedia controls allows clients of the Web API to explore and navigate through resources without being tightly coupled to the internal characteristics of the implementation.

B. CoHA Maturity Model

Classification of HTTP-based APIs (CoHA) [4] defines five maturity levels that can be associated with the design of Web APIs, as shown by Fig. 1(b). The first level, named WS-*, corresponds to implementations based on the SOAP protocol that employ HTTP as data transfer mechanism. In this level, URIs are associated with services, and SOAP envelopes encapsulate the required information for service execution. This strategy presents high cost of adoption, maintenance and evolution, and may result in vendor lock-in due to the use of proprietary technology.

The second maturity level, named RPC URI-tunneling, requires the use of resources for modeling the API, without enforcing operations based on the semantics of the data transfer protocol. Despite allowing resources to be uniquely identified, this level does not grant access to multiple resource representations. This design approach results in high coupling between client and server, and constrains their capacity to evolve independently from each other.

The third level, called HTTP-based Type I, requires handling resources according to the semantics of the HTTP protocol. The API provides access to resources through representations, allowing the use of different data formats. This design requires the server to present stateless behavior, increasing the scalability of the Web API. It has a low cost of adoption, but the resulting implementation is still hard to evolve due to the high coupling between client and server.

The fourth level is named HTTP-based Type II and requires the use of self-described messages. Therefore, operations and resources must be described at design time. Web APIs that adopt this design strategy simplify client-server interactions by providing a uniform interface.

The fifth and highest level is called REST and follows all the constraints define by this architectural principle (i.e., resources are identified and addressed, handled through representations, messages are self-described and provide hypermedia controls). A high

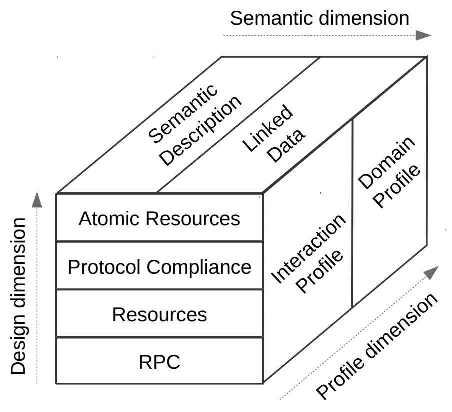

Figure 2. The WS^3 maturity model

learning curve is required for adopting this design strategy, but this initial cost is compensated by lower maintenance and evolution costs, because changes in the Web API have less impact on the client.

IV. The WS^3 Maturity Model

The WS^3 maturity model, as shown in Fig. 2, has a cubic shape, in which the faces represent the evolution of Web APIs according to three dimensions, identified as *Design*, *Profile* and *Semantics*. The *Design dimension* is based on the Richardson Maturity Model, described in section 3.1, and represents the different modeling strategies adopted for designing a Web API. The *Profile dimension* reflects the quality of documentation provided by the Web API. The *Semantics dimension* describes the level of support for semantic description of data and operations presented by the Web API. These three dimensions are independent from each other and can be freely combined to classify a given implementation.

According to Chahal and Singh [16], quality is a measure of user satisfaction, which is perceived by users based on external attributes of the system. Some outstanding external attributes are: usability, integrity, maintainability and reliability. However, internal attributes - i.e., cohesion, coupling and reusability - are of major relevance to the development team. The WS^3 maturity model sees client applications as users of the Web API, and establishes its internal and external characteristics to fulfil client expectations. In order to reach this goal, quality factors, metrics and criteria are distributed across the three dimensions of the WS^3 model.

A. The Design Dimension

The *Design dimension* evaluates the structural characteristics of a Web API. This dimension focuses mainly on the structure of the information provided by the API. The *Design dimension* has four maturity levels. The lowest maturity level is assigned to APIs with RPC design, which are strongly based on services, and therefore do not comply with REST principles. The *Resources* level corresponds to implementations in which data provided through the API is modeled in the form of resources. This design cohesively groups data into separate units and simplifies information handling, allowing the definition of operations for managing data. Up from this level, APIs must have stateless behavior, resources are handled through their representations and messages are self-described. The following level, named *Protocol Compliance*, is associated with Web APIs that respect the semantics of the data transfer protocol. From this level upwards, resources are handled through a uniform interface imposed by the adopted protocol. In spite of the focus of the *Design dimension* being on the structural characteristics of the system, the *Protocol Compliance* level evaluates behavioral aspects that aim to enforce communication patterns. At the top of this face of the cube is the *Atomic Resources* level, which corresponds to the most mature design strategy. This level constrains access to resource properties separately. In this case, the smallest data unit that can be handled by operations is the resource, and the provided operations are able to alter all properties contained in a representation. The *Atomic Resources* level is directly linked to information integrity, given that the constraints imposed by this design approach increase this quality criterion.

Fig. 3 presents an example of atomic design, in which two separate resources represent a single person. By adopting this design it is possible to handle part of data associated to a person separately, guaranteeing the atomicity of operations. Fig. 4 illustrates an example with a representation of a person. The resource is composed by the following properties: *FirstName*, *FamilyName*, *BirthDate*, *Email* and *Password*. CRUD operations can be performed on the resource through HTTP GET, POST, PUT and DELETE messages. However, the resource provides a second set of operations to handle only properties *Email* and *Password*. Despite respecting the semantics of the HTTP protocol, the adopted design does not guarantee the atomicity of operations performed on this resource.

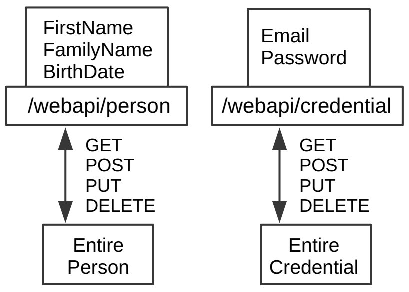

Figure 3. Example: Atomic resource.

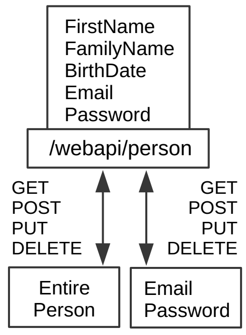

Figure 4. Example: Non-atomic resource.

B. The Profile Dimension

A profile provides details on the structure and behavior of resources handled through a Web API. Self-documentation is one of the criteria presented by Meskens [14] that is directly related to the three quality factors (i.e., flexibility, maintainability and testability). The profile is an effective way for self-documenting an API. Therefore, APIs that reach the maturity levels associated with the *Profile dimension* have this quality characteristic. The proposed maturity model takes into account only profiles that can be interpreted by software agents. This constraint reinforces the premise that the documentation must be employed only for guiding clients at runtime, instead of guiding the developer (i.e., a human) at design time.

The *Profile dimension* can be evaluated combined with the other dimensions (Design and Semantic), and is divided into two maturity levels. Web APIs classified in the maturity level *Interaction Profile* provide documentation describing the semantics of the adopted communication protocol. This maturity level requires the description of all operations, including their execution details, such as HTTP method, URI, supported media types, required properties, and so on. The *Domain Profile* level requires the description of details specific to the application domain of the Web API, such as the order of operation execution, pre- and post-conditions, etc. This level may be reached using three different strategies: adding hypermedia controls to resource representations; through protocol headers; or by documenting the Web API. Meskens [14] considers that systems can be seen based on different levels of abstraction. The *Interaction* and *Domain Profile* levels of the WS^3 model correspond, respectively, to the functional and domain abstraction levels defined by Meskens.

It is important to notice that complex business processes may require several requests to manipulate multiple resources. A possible strategy to hide the complexity from the user is to encapsulate all resources required to execute a complex process in a single meta-resource, allowing the execution of several operations with a single request. This approach is recommended if the client application has previous knowledge of all the information needed for executing the whole process. Some processes, though, require information produced or provided by the Web API during the execution of intermediate steps of the business process. In this scenario, the process cannot be invoked with a single request; therefore, a mechanism that guides the client through the execution of a sequence of requests is required. Hypermedia controls added to resources may be employed to guide the client, teaching how to execute operations on resources. This information can be added to resource representations or to protocol headers. Fig. 5 shows an example in which a resource *Person* is created through an HTTP POST request. The Web API sends a response saying that the resource was successfully stored and informing the next step, which is creating a *Credential* for this person.

The strategy based on hypermedia controls provides a mechanism for orchestrating operations that is not tied to a given application domain, given that the execution of business processes is directly based on resources and operations. Despite the fact that this mechanism can be used in any application domain, there is still a high structural coupling among resources handled by the Web API. This structural coupling results from the fact that

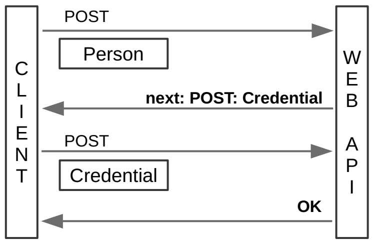

Figure 5. Orchestration through hypermedia control

client applications must correctly interpret resource representations, which may present ambiguities, leading to handling errors. A viable alternative is to describe resource semantics by associating properties with concepts described by a vocabulary or ontology.

C. The Semantic Dimension

Web APIs described through semantic technology give an increased understanding of the way in which resources are represented and consumed. With the Semantic Web, information can be consistently handled, avoiding all sorts of ambiguity. The semantic description of resources can be done through the association of concepts defined by vocabularies shared with client applications. Through this approach, vocabularies become the knowledge base, reducing the coupling among Web APIs and their clients to the conceptual level. The *Semantic dimension* of the maturity model represents the use of semantic technology and is divided into two maturity levels, which can be combined with the other two dimensions (i.e, Design and Profile).

Heitmann [2] highlights that the adoption of the Semantic Web for system development is in expansion, and that one of the main reasons for its adoption is the ability to simplify system integration. Despite the increased adoption of semantic technologies, a large share of system integration efforts require human intervention. Heitmann states that one of the main challenges in this area is to integrate different sources of heterogeneous data. The Semantic Web can provide a solution for executing complex data integrations. Due to these facts, the *Semantic dimension* of the WS^3 maturity model is an essential part in the process of classifying and evaluating Web APIs.

The *Semantic Description* level corresponds to APIs in which properties and operations of resources are semantically described, allowing consumers to understand the information and to perform operations appropriately (Fig. 6). Web APIs that provide semantic description of resources result in less coupled applications, given that the consumer can take into account the semantic descriptions instead of relying on the developer to hard-code his/her understanding of the application domain. However, the semantic description of properties and operations of resources incurs in a high cost, due to the high modeling and development complexity.

The *Linked Data* level is reached when not only resources, but also the relationships amongst them are semantically described. Despite being possible to describe relationships through the documentation of the Web API, the use of hypermedia controls results in a more beneficial solution, which allows consumer applications to obtain this information on demand [13][5].

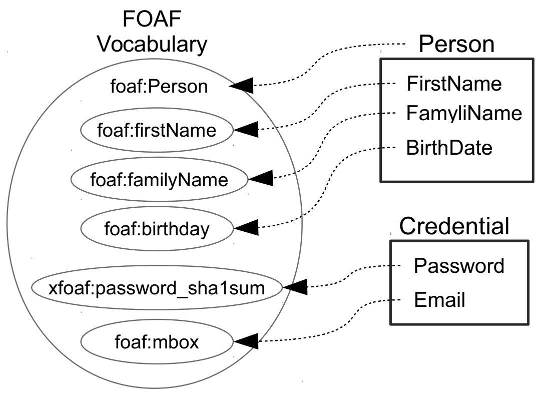

Figure 6. Semantic description

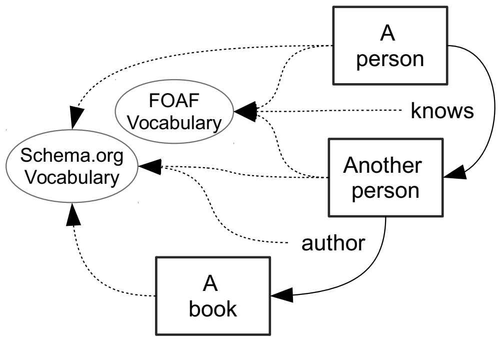

Figure 7. Example: Linked Data

```
{
  "@context": {
    "foaf": "http://xmlns.com/foaf/0.1/",
    "schema": "http://schema.org/"
  },
  "@type": ["foaf:Person","schema:Person"],
  "foaf:firstName": "A",
  "foaf:familyName": "person",
  "foaf:knows": {
    "@type": ["foaf:Person","schema:Person"],
    "foaf:firstName": "Another",
    "foaf:familyName": "person",
    "schema:author": {
      "@type": "schema:Book",
      "schema:name": "A book"
    }
  }
}
```

Figure 8. Example: Resource representation in JSON-LD

Fig. 7 presents an example of use of Linked Data, in which three resources are semantically described by two different vocabularies that have equivalent concepts. If more vocabularies are employed to describe resources, higher is the chance of a client application being able to correctly understand the information carried by the resource. The capability of interpreting information is even higher when vocabularies are shared by several other applications. This occurs with some public vocabularies such as *FOAF*, *Schema.org*, *DublinCore* and *DBPedia*, among others. However, a proprietary vocabulary can be employed to describe a very specific domain that is not completely described by an existing public vocabulary. Besides describing the semantics of properties, the example shown in Fig. 7 identifies relationships among resources, which are also defined by controlled vocabularies. One can notice in the example that *A person* knows *Another person* that is the author of *A book*. The use of Linked Data allows performing inferences that result in a higher level of information reuse. It is possible, for example, to infer that A person knows a book writer. The Linked Data maturity level does not imply using a standard data format, but Web APIs can adopt standard technologies such as RDF and JSON-LD, among others, as shown by Fig. 8.

D. Classification and Selection of Web APIs

The WS^3 maturity model provides classification mechanisms that are helpful for the selection and composition of Web APIs. However, given that the maturity model has three dimensions that can be freely combined, the maturity level of a given Web API cannot be represented by a single value. Instead, a triplet must be used to represent and compare maturity levels. This triplet may be employed for API selection, but the capabilities and requirements of the client application must also be taken into account.

Fig. 9 shows an example in which a client application must invoke a service provided by a Web API. Multiple APIs provide similar services, which are implemented differently. The client application must choose among the available Web APIs based on some criteria. The WS^3 maturity model can help in this process, identifying the maturity level of each existing Web API. As shown in Fig. 9, the *Simple Web API* reaches levels *D3-S0-P0*, meaning that it fits in the third level of the Design (D) dimension (i.e., *Protocol Compliance*) and that it does not provide support for Semantics (S) and Profile (P). By comparing this result with the levels obtained by the other APIs, the client application can choose the one that is more suitable for its needs. The chosen API, though, might not be the one that reaches the higher maturity level in all dimensions. The benefits provided by a more mature API are not always useful for the client. In addition, reaching a higher maturity level often results in more resource consumption and in performance loss, impacting negatively on client behavior.

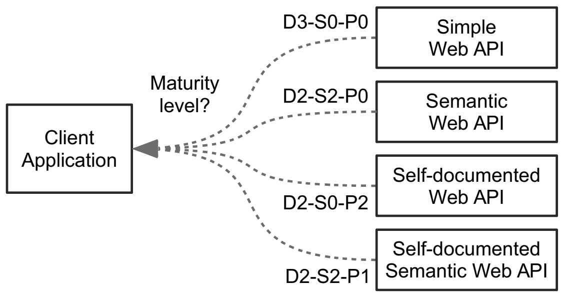

Figure 9. Selection of Web API based on maturity level

E. Comparative Analysis

Richardson and CoHA maturity models provide very useful means for classifying the design of Web APIs. These maturity levels allow designers to identify the desired characteristics of a Web API and guide its development and evolution. Despite providing a useful reference for evaluating the quality of Web API design, the maturity model proposed by Richardson does not describe ways to reach the highest maturity levels [19] and takes into account a limited set of quality attributes. On the other hand, CoHA provides a more comprehensive alternative for Web API evaluation, which is not constrained to RESTful systems, taking into account other HTTP-based distributed applications. CoHA also evaluates attributes such as performance, visibility, cost, evolution and maintenance, among others. However, CoHA classifies RESTful Web APIs in a single level that requires full adherence to REST principles.

The WS^3 maturity model, which was introduced in this paper, takes into account characteristics that are not considered by Richardson and CoHA, such as the use of profiles and semantic descriptions. Profiles are vital for documenting Web APIs that deal with complex business processes, while the use of semantic descriptions helps clients of the Web API to understand and handle data correctly. The evaluation done by Heitmann [2] shows that only 11% of applications that employ Semantic Web technology execute a fully automated integration process. Given that the *Profile dimension* considers only documentation that can be read by software agents, the documentation of Web APIs can be used together with semantic technology to allow application integration without human intervention. The WS^3 maturity model may be used as a mechanism to promote automatic usability and composability. In order to allow such degree of automation, a Web API has to reach high maturity levels in different dimensions of the model, given that integration without human intervention requires features present in different dimensions.

Richardson and CoHA maturity models do not clarify how the highest maturity levels can be reached. The WS^3 maturity model, on the other hand, suggests the use of hypermedia controls, Linked Data and profiles to obtain high-quality API design. The *HTTP-based Type I* level of the CoHA model requires using Web Application Description Language (WADL) [20] for describing resource representations and operations available through the Web API. WADL may be seen as an approach for implementing profiles, however it is not able to describe the orchestration of requests required for executing a complex business process.

V. Evaluation Of Support Technologies For Web API Development

According to Heitmann [2], the main obstacles for developing semantic applications are: integrating noisy data; mismatched data models between components; distribution of application logic across components; and missing guidelines and patterns. Maturity models can be employed as patterns and guidelines for Web API development, aiming to increase the quality of the resulting product. Software development tools and frameworks can also take into account the evaluation and classification criteria identified by maturity models in order to anticipate and mitigate obstacles during system development, evolution and maintenance. This section evaluates some tools and frameworks that provide support for Semantic Web API development.

Table I
Maturity Level Of Support Technologies For Web API Development

Support Technology	Dimension		
	Design	Profile	Semantic
Maleshkova et al. [21]	All levels	None	None
Gulden & Kugele [23]	All levels	None	None
Strauch & Schreier [19]	All levels	Interaction	None
Salvadori & Siqueira [24]	All levels	Interaction	Linked Data

The vast majority of Web APIs are described only through text-based documentation, forcing users to read and understand these documents to write client applications. This fact results in high-coupled implementations that are difficult to reuse. In this context, Maleshkova et al. [21] propose an approach to semantically annotate and classify the documentation of Web APIs. The proposed approach employs the SWEET Tool [22] to add semantic descriptions to Web API documentation through the execution of a semi-automated process.

Gulden and Kugele [23] propose an approach to add RESTful Web interfaces to legacy systems through the automatic generation of Java code based on models. It generates resources based on the Domain Transfer Object (DTO) pattern, which are composed only by properties. The integration code is isolated in a separate layer containing service classes. This approach provides partial support for hypermedia control.

Strauch and Schreier [19] propose a procedure composed by three steps for translating SOAP-based service interfaces into a RESTful Web API. The first step comprises the conversion of services into resources, which are handled through POST operations (RPC style). The second iteration applies a uniform interface with CRUD operations, which are invoked through the corresponding HTTP request messages. The third step adds hypermedia controls, defining the relationships among different resources. Support for HATEOAS is obtained through URI templates.

Salvadori and Siqueira [24] propose a framework that allows resources to be annotated with semantic descriptions. The framework is based on the JAX-RS specification, which provides support for the development of RESTful Web Services on the Java platform. Annotations are added to resource properties and operations directly in the source code of the Web API. These annotations result in resource representations in JSON-LD format with hypermedia controls based on Hydra [12]. The framework generates the documentation of the API based on information obtained from the annotations added to the source code.

Table 1 compares the maturity level of the support technologies for Web API development described in this section, taking into account the three dimensions of the maturity model proposed in this paper. All the described technologies allow reaching any level in the *Design dimension*, given that they do not have a direct influence on the design principles adopted for building the Web API. Only [24] and [19] reach *Interaction* level in the *Profile dimension*, and none reaches the *Domain* level. Just [24] provides full support for Semantic description of the Web API and its resources. Semantic annotations are added to the Web API documentation by [21], but it does not add semantic information to resource representations.

VI. Conclusions

The WS^3 maturity model aims to guide the development of distributed applications that employ the Web as communication infrastructure. The resulting model emphasizes the relevance of efficient design as well as the provision of descriptive information to obtain integration APIs that result in implementations with low coupling and able to correctly understand the semantics of exchanged data.

The proposed maturity model helps to identify the main goals to promote healthful interaction among clients and the Web API. First, the API must expose a data model based on atomic resources and the provided operations must respect protocol semantics. The Web API must also be described using profiling mechanisms, allowing software agents to understand how to interact with resources. At last, the use of Semantic Web technologies to describe resource representations accessed through the Web API allows clients to dynamically find out how data is represented, which operations are available and how resources relate to each other.

As seen in section V, the existing support technologies for Web API development still do not promote high quality design, and have very limited support for building profiles and for semantic description of resources. It is expected that a new generation of support technologies looks into these limitations, aiming to improve the quality of the resulting Web APIs.

References

[1] D. I. K. Sjoberg, T. Dyba, and M. Jorgensen, "The future of empirical methods in software engineering research," ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 358–378. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.30

[2] B. Heitmann, R. Cyganiak, C. Hayes, and S. Decker, "An empirically grounded conceptual architecture for applications on the web of data," *Trans. Sys. Man Cyber Part C*, vol. 42, no. 1, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1109/TSMCC.2011.2145370

[3] A. Tahir and S. Macdonell, "A systematic mapping study on dynamic metrics and software quality," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, Sept 2012, pp. 326–335.

[4] J. Algermissen, "Classification of HTTP-based APIs," 2010. [Online]. Available: http://nordsc.com/ext/classification_of_http_base_apis.html

[5] J. Webber, S. Parastatidis, and I. S. Robinson, *REST in Practice - Hypermedia and Systems Architecture.* O'Reilly, 2010.

[6] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, May 2001. [Online]. Available: http://www.scientificamerican.com/article/the-semantic-web

[7] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee, "Linked data on the web (ldow2008)," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 1265–1266. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367760

[8] W3C, "Resource Description Framework (RDF) Model and Syntax Specification," 2014. [Online]. Available: http://www.w3.org/TR/rdf-syntax-grammar/

[9] R. T. Fielding, "REST: architectural styles and the design of network-based software architectures," Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[10] L. Richardson, M. Amundsen, and S. Ruby, *Restful Web Apis*. Oreilly & Associates Incorporated, 2013. [Online]. Available: http://books.google.com.br/books?id=i3a7mAEACAAJ

[11] M. Lanthaler and C. Gütl, "On using JSON-LD to create evolvable RESTful services," in *WS-REST*, R. Alarcón, C. Pautasso, and E. Wilde, Eds. ACM, 2012.

[12] M. Lanthaler, "Creating 3rd Generation Web APIs with Hydra," ser. WWW '13 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 35–38.

[13] I. Robinson, "RESTful Domain Application Protocols," in *REST: From Research to Practice*, E. Wilde and C. Pautasso, Eds., 2011, pp. 61–91.

[14] N. Meskens, "Software quality analysis system: a new approach," in *Industrial Electronics, Control, and Instrumentation, 1996., Proceedings of the 1996 IEEE IECON 22nd International Conference on*, vol. 3, Aug 1996, pp. 1406–1411 vol.3.

[15] N. Nik Daud and W. Wan Kadir, "Static and dynamic classifications for SOA structural attributes metrics," in *Software Engineering Conference (MySEC), 2014 8th Malaysian*, Sept 2014, pp. 130–135.

[16] K. Kaur Chahal and H. Singh, "A metrics based approach to evaluate design of software components," in *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, Aug 2008, pp. 269–272.

[17] H. Zuse, *Software Complexity: Measures and Methods*. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1991. [Online]. Available: http://dl.acm.org/citation.cfm?id=102702

[18] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *Software Engineering, IEEE Transactions on*, vol. 35, no. 5, pp. 684–702, Sept 2009.

[19] J. Strauch and S. Schreier, "RESTify: From RPCs to RESTful HTTP Design," in *Proceedings of the Third International Workshop on RESTful Design*, ser. WS-REST '12. New York, NY, USA: ACM, 2012, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/2307819.2307824

[20] M. J. Hadley, "Web Application Description Language (WADL)," Sun Microsystems, Inc., Mountain View, CA, USA, Tech. Rep., 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1698142

[21] M. Maleshkova, L. Zilka, P. Knoth, and C. Pedrinaci, "Cross-Lingual Web API Classification and Annotation," in *MSW*, ser. CEUR Workshop Proceedings, E. Montiel-Ponsoda, J. McCrae, P. Buitelaar, and P. Cimiano, Eds., vol. 775, 2011, pp. 1–12.

[22] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Semantic annotation of Web APIs with SWEET," in *6th Workshop on Scripting and Development for the Semantic Web, colocated with ESWC*, 2010.

[23] M. Gulden and S. Kugele, "A Concept for Generating Simplified RESTful Interfaces," in *Proceedings of the 22nd International Conference on World Wide Web Companion*, 2013.

[24] I. Salvadori and F. Siqueira, "A Framework for Semantic Description of RESTful Web APIs," in *Web Services (ICWS), 2014 IEEE 21th International Conference on*, 2014, pp. 630–637. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2014.93