Investigating Web APIs on the World Wide Web

Maria Maleshkova, Carlos Pedrinaci, John Domingue
Knowledge Media Institute (KMi)
The Open University
Milton Keynes, United Kingdom
{*m.maleshkova, c.pedrinaci, j.b.domingue*}*@open.ac.uk*

***Abstract*—The world of services on the Web, thus far limited to "classical" Web services based on WSDL and SOAP, has been increasingly marked by the domination of Web APIs, characterised by their relative simplicity and their natural suitability for the Web. Currently, the development of Web APIs is rather autonomous, guided by no established standards or rules, and Web API documentation is commonly not based on an interface description language such as WSDL, but is rather given directly in HTML as part of a webpage. As a result, the use of Web APIs requires extensive manual effort and the wealth of existing work on supporting common service tasks, including discovery, composition and invocation, can hardly be reused or adapted to APIs. Before we can achieve a higher level of automation and can make any significant improvement to current practices and technologies, we need to reach a deeper understanding of these. Therefore, in this paper we present a thorough analysis of the current landscape of Web API forms and descriptions, which has up-to-date remained unexplored. We base our findings on manually examining a body of publicly available APIs and, as a result, provide conclusions about common description forms, output types, usage of API parameters, invocation support, level of reusability, API granularity and authentication details. The collected data provides a solid basis for identifying deficiencies and realising how we can overcome existing limitations. More importantly, our analysis can be used as a basis for devising common standards and guidelines for Web API development.**

***Keywords*-Web APIs, RESTful services, Web services**

I. Introduction

The world of services on the Web is increasingly dominated by Web applications and APIs, which seem to be preferred over "classical" Web services based on WSDL and SOAP. Web services have played and, without a doubt, will continue to play a major role for the development of loosely-coupled component-based systems within and between enterprises. However, Web APIs, also referred to as RESTful services [1] when conforming to the REST architectural principles [2], are characterised by their relative simplicity and their natural suitability for the Web, relying almost entirely on the use of URIs, for both resource identification and interaction, and HTTP for message transmission. On the basis of this simple technology stack, many Web sites like Facebook, Google, Flickr and Twitter offer easy-to-use, public APIs that provide simple access to some of the resources they hold, thus enabling third-parties to combine and reuse heterogeneous data coming from diverse services in data-oriented service compositions called mashups.

Despite their popularity, the use of Web APIs still requires extensive manual effort, which is most often focused on the development of custom tailored software that can hardly be reused. A number of researchers and developers are devising generic solutions for better supporting the discovery, reuse, invocation, and composition of Web APIs [3], [4]. These approaches build upon the wealth of research on Web services and adapt it to deal with Web APIs. Yet, a quick look at some of the existing Web APIs shows significant differences when compared to classical Web services. The most notable distinction lies in the fact that there is no established interface definition language, although some researchers have already tried to address this aspect [5], [6]. In fact, as opposed to Web service technologies, work around Web APIs has evolved in a rather autonomous way, which is perhaps one of the main reasons for their rapid proliferation.

Before any significant impact and improvement can be made to current Web API practices and technologies, we need to reach a deeper understanding of these. This involves, for instance, figuring out how current APIs are developed and exposed, what kind of descriptions are available, how they are represented, how rich these descriptions are, etc. It is only then that we shall be able to clearly identify deficiencies and realise how we can overcome existing limitations, how much of the available know-how on Web services can be applied and in which manner.

To this end, in this paper we present a thorough analysis over a body of publicly available API descriptions. In particular, we analyse how Web APIs are published, we check which information is provided and its level of detail. We investigate the characteristics of input parameters and record the API categories. Similarly, we study the provided output descriptions and analyse the different types of APIs as well as the availability of relevant details such as the HTTP method, invocation URI and authentication requirements. We also record whether example requests and responses are provided, since they indicate how the communication between the client and the server is realised. Finally, we also study general API information, such as the number of mashups and operations, in order to be able to draw conclusions about the reusability and the granularity of the

APIs. The analysis exposed in this paper provides a reality check over the current state and practices with Web APIs and certainly contributes to understanding where we are, helps us in better realising what needs to be done, and also assists us in devising supporting mechanisms. In this sense, we show that the current proliferation of Web APIs is not due to the increased use of REST principles, since according to our study, most Web APIs do not have RESTful descriptions and how APIs are described is not significant for reusability. Instead, simplicity and the trend towards opening data are driving the evolution that results in the world of services on the Web being increasingly dominated by Web applications and APIs.

The remainder of this paper is structured as follows: Section II, describes the methodology used for conducting our Web API study, while Section III gives the collected data. A summary of the main results and a discussion of identified correlations and trends are provided in Section IV and Section V. Section VI presents an overview of existing work on analysing Web services and Section VII presents future work and concludes the paper.

II. Methodology

The study presented herein was conducted during February 2010, analysing 222 Web APIs from the ProgrammableWeb[1] directory. ProgrammableWeb is a popular API directory, that at the time of this writing provides information about 2002 APIs and 4827 mashups. For easier search and browsing, the APIs are sorted in categories and our analysis covered all 51 categories, including on average 4 APIs per category. The analysed Web APIs for each category were randomly chosen, however, since some categories have only one or two entries, the analysed number of Web APIs per category varies. As a result the survey covered 18% of the REST APIs listed at ProgrammableWeb (1235 APIs at the time of the study). Therefore, we consider the following results to be representative for the directory and in general, since ProgrammableWeb is currently the biggest directory[2].

Each Web API description was analysed in terms of six main groups of features, including general Web API information, type of Web API, input parameters, output formats, invocation details and complementary documentation. The Web API analysis was conducted manually, and some features such as the *type of Web API* were examined twice in order to achieve greater accuracy. More concretely, each Web API was examined in terms of:

1) General Web API information – name of the API, description, category, number of mashups, date updated, URL and number of operations.
2) Type of Web API – details on whether the API description is RESTful, RPC-style or hybrid (for more details see section III-B).
3) Input parameters – does the API use default parameters, does it use optional parameters, does it use coded parameters (for example, instead of ”English“ use ”en“), does it use parameters with alternative values (for example, the input value is 1 or 2 or 3), is the data-type of the input parameter stated and are boolean (yes/no, true/false) parameters used.
4) Output formats – form of the output (for example, XML or JSON) and whether it is sent as a parameter.
5) Invocation details – is the HTTP method provided, is the invocation URI provided, does the API require authentication and if yes, what type, how are the input parameters transmitted and how is the authentication information transmitted.
6) Complementary documentation – does the description provide example request, example response and a list of error messages/codes.

We focus our analysis on studying precisely these groups of API features because each of them plays an important role for different aspects of the API use. The general information provides insights on the information that is commonly used to describe Web APIs in directories and how this information is captured, including temporal details, reusability and level of granularity. Since, an important part of current research work on APIs is focused on investigating and opposing different Web service types (REST vs. WSDL and SOAP) [3], we also record and analyse the existing types of Web APIs. We study input parameters, output formats and invocation details, since they serve as the basis for conducting main service tasks. These Web API features are present in all interface description languages (IDLs), as they are considered essential for invocation [3], composition [4] and discovery. The complementary documentation provides details on how the communication between the client and the service is realised, and what are the possible errors that can occur.

The analysis approach involved a sequence of simple steps. First, for each API picked for the study, the ProgrammableWeb webpage was opened. The APIs to analyse were randomly chosen within each category, covering all categories. This was necessary in order to ensure that the results are domain-independent and at the same time representative for the whole directory. For each API the general information was recorded.

Second, the provider's Web API description was examined, recording the documentation URL, counting the number of operations and determining the type of the API. For RPC-style and hybrid operations each operation was counted, while for RESTful ones, each resource representation manipulation/retrieval through an HTTP method was counted as one operation. For example, GET on the *User-*

[1]http://www.programmableweb.com

[2]Webmashup.com (http://www.webmashup.com) contains around 1800 Web APIs and 3100 mashups, while APIFinder (http://www.apifinder.com) provides around 1100 Web APIs.

Profile resource is one operation, while PUT on the same resource is another. We also analysed the input parameters of each operation, in case of RESTful services these are also referred to as the *scope* [1]. For the output of each API, the format was recorded, including the available alternatives and how they are chosen (through parameterisation or through a separate URI for the invocation). Finally, the invocation details, included in the description, and the complementary documentation were recorded. We did not perform any test invocations of the APIs, since we aim to gain a picture of the current state of the Web APIs landscape as depicted by their descriptions.

Conducting the study took around three weeks, since the documentation of every Web API had to be review manually. In the process, we already noticed that the work was slowed down by the fact that the description forms and structures are very diverse and each API had to be examined from scratch, without being able to benefit from the analysis of previous APIs. This already provides some indication about the difficulties arising from having to deal with heterogeneous textual API documentation.

III. Analysing Common Web API Descriptions

In this section we describe the data collected from the Web API study. The results are structured into six groups, according to the different parts of the API descriptions that were analysed.

A. General Web API Information

The general Web API information analysed includes the recording of some details provided directly by the API directory, such as the name of the API, its description, the category that it is assigned to, the URI of the API and the latest update of the description. Table I provides the exact numbers for these features.

Table I: General Web API Information

Description	Maximum	Minimum	Average
APIs per Category	12	1	4
Number of Mashups	506	0	6.4
Number of Operations	over 200	1	15.5

Of these general details, the number of mashups is of particular relevance, since it provides an indicator of the reuse of Web APIs, and to a certain extent can help to highlight factors influencing the reusability of APIs. The analysis shows that a few APIs are highly reused, whereas most APIs, are used in very few or no mashups at all. In particular, there are 136 APIs with 0 mashups, 60 APIs with 1 to 4 mashups and 26 APIs with 5 to 506 mashups. The API with most mashups is Flickr, which can be easily integrated into different Web applications as a source of images and photos. In summary, there is a big difference in the frequency of use of some APIs (12%), while most APIs are not used often as part of mashups. Also it must be noted, that the number of mashups is as provided by ProgrammableWeb, therefore the actual values can somewhat differ. However, for comparison purposes it is still representative, since the data comes from the same source for all APIs.

The general API information collected also delivers some valuable insights about the granularity, i.e. the number of operations, of the APIs. 109 of the APIs or about 50% have 1 to 7 operations, while 36 APIs or 16% have only 1 operation. 92 APIs have between 7 and 50 operations, where more APIs have fewer operations. Finally, only 21 APIs have between 50 and 200+ operations (Yahoo Ads). This leads us to the conclusion that the majority of the APIs are small and have very few operations. We investigated whether there is a correlation between the size of the APIs and their use as part of mashups, but even though social and community Web sites, seem to expose a larger number of operations, there are important exceptions such as del.icio.us[3], which has only 15 operations but 142 mashups and geocoder[4] with 3 operations but 28 mashups. The data provided no proof that there is a relation between the level of reusability of APIs and their granularity.

The analysed API descriptions were updated between 02.06.2005 and 14.01.2010, which shows that the ProgrammableWeb directory has been enriched during the past five years, but also that some descriptions are old, which might be an indication that they are out of date. There were relatively few descriptions from 2005, 2006 and 2007 (11, 33, 27 correspondingly) and around 60 and 80 for 2008 and 2009. This might indicate either that there have been an increasing number of APIs published during the past two years or that older descriptions have been update, even though the APIs were created earlier. Since we do not have the date of creation of the API entries but only the update dates, we cannot make a conclusive statement.

Finally, based on the general Web API information, our analysis highlighted that since all details are added manually to the Web API directory, some of the feature descriptions were not always accurate. This is especially true for the URL of the documentation, which was sometimes moved or no longer available, and for the authentication information, which was very often inaccurate. This is indicative for the difficulties resulting from using directories based on user entries, the two main ones being the retrieval of outdated information, because the entries cannot be automatically updated, and the retrieval of erroneous information, due to wrong or inaccurate user input. Therefore, despite the fact that currently these manually created directories are the easiest way to search for APIs, there is a need for developing approaches that automatically crawl and extract accurate API descriptions from the Web.

[3] http://delicious.com/help/api
[4] http://geocoder.us/help/

B. Type of Web APIs

In this section we describe our findings regarding the different types of Web APIs and their frequency of use. We have identified three types of APIs: RESTful, RPC-style and Hybrid. RESTful services are defined as services, which conform to the representational state transfer (REST) paradigm [2]. REST is based on a set of constrains such as the client-server based communication, statelessness of the request and use of a uniform interface. A RESTful web service is commonly implemented by using HTTP, comprising a collection of uniquely identified resources and their links to each other. In addition, RESTful services are characterised by resource-representation decoupling, so that resource content can be accessed via different formats.

For the scope of our study, we identify Web APIs as RESTful, when their descriptions indicate that they are resource-centred and data retrieval and manipulation is done only over the HTTP methods. Example APIs include MusicBrainz (http://wiki.musicbrainz.org/XMLWebService) and Doodle (http://doodle.com/xsd1/RESTfulDoodle.pdf). RESTful services can have a scope, or a set of parameters, to restrict the effect of the HTTP methods on the resource only to the ones determined by the parameter values. For example, instead of retrieving all news resources in the *News* collection by using GET (HTTP GET http://url/.../News) the API can also be invoked by including a parameter and retrieve only news created by a particular user (HTTP GET http://url/.../News?user=aUser).

In comparison to RESTful APIs, RPC-style ones do not use directly the HTTP methods to access resources but rather define their own operations, wrapping the resource information, and then invoke these through one of the HTTP methods. For example, an RPC-style API, providing the same information as the news RESTful one, would look like: HTTP GET http://url/.../getNews and there can be a scope or a set of parameters (HTTP GET http://url/.../getNews?user=aUser). Example APIs include GeoNames (http://www.geonames.org/export/web-services .html) and Daylife (http://developer.daylife.com/docs). It is important to point out that we base our classification strictly on the API descriptions, since RPC API implementations can be wrapped and described as RESTful and RESTful implementations can have operations such as *getNews*, which are in fact realised by using the GET HTTP method on the *News* resource. Still our definitions of Web API types share common understanding with the ones given in [1], stating in essence that RPC APIs exposes internal functionalities through a complex programming-language-like interface that is different for every service, while resource oriented APIs exposes internal data through a simple document-processing interface that is always the same.

Hybrid APIs, as the name suggests, represent a mix between RESTful and RPC ones. Hybrid-style APIs define their own operations, but employ operation information, which is contradictory to the used HTTP method. For example, a hybrid API can realise the *getNews* operation through POST and *addNews* trough GET. Example hybrid APIs include ClearForest (http://www.opencalais.com/documentation/calais-web-service-api), which uses POST for getting resources and Box.net (http://developers.box.net/ApiOverview) where adding a new element can be done by using GET. The use of hybrid APIs can be very problematic since they do not guarantee operation safety, especially in cases where data manipulation is realised by using GET, because of the possibility of unintentional data modification. In such cases a simple crawler can change or delete resources, since it would use GET, expecting to retrieve information instead of altering it.

Table II: Type of Web APIs

Description	In %
RPC-Style	47.8
RESTful	32.4
Hybrid	19.8
Mashups with RPC-Style APIs	42
Mashups with RESTful APIs	34
Mashups with Hybrid APIs	24

Table II shows the distribution of the different types of APIs. As it can be seen, currently almost half of the Web APIs are RPC-style and about one third are RESTful. The hybrid APIs represent about 20% of the analysed data. This shows that even though RESTful services are by design suitable for the Web, since they are based on the same principles, their level of adoption is still relatively low. Instead of identifying resource collections and manipulating them with the help of HTTP methods, developers prefer to define their own operations, whose functionality sometimes even contradicts the used HTTP method (hybrid APIs). As a result, two thirds of the API descriptions are structured very much like common interface definitions, disregarding the REST principles.

A very similar distribution can be detected among the APIs, which are reused as part of mashups. 42% of the APIs are RPC-style, 34%– RESTful and 24%– hybrid. Therefore we can conclude that API reuse is not driven by the type of description, since the mashups percentage distribution matches almost exactly the Web API distribution. As a result, we can argue that the current proliferation of Web APIs cannot be attributed to the use of RESTful services. As our study shows, most Web APIs do not have RESTful descriptions and how APIs are described is not significant for reusability.

C. Input Parameters

We also thoroughly analysed the information in the API descriptions, relating to the input parameters. As it can be seen in Table III about 60% of APIs use optional parameters,

while 45% use default values. This has a strong effect on the matchmaking and invocation approaches, since one API can be found or not depending on whether optional parameters are taken into account or not. Similarly, if invocation is done on the basis of default values, the output results can be drastically changed. For example, a lot of APIs have XML as a default output format but some use also JSON as default. If the default parameter value is used, the results might be retrieved in the wrong format, making them useless.

Table III: Input Parameters

Description	Number	In %
APIs w/t optional parameters	136	61.3
APIs w/t alternative values for a parameter	114	51.3
APIs w/t default values for parameters	99	44.6
APIs that state the data-type of the parameters	61	27.5
APIs w/t coded values for a parameter	55	24.8
APIs w/t boolean parameters	39	17.6

The fact that a lot of APIs use alternative values for one parameter (for example, a range of 1, 2 or 3) and coded values (for example, for languages only a language code, instead of the full string) makes the API invocation even more challenging. For the invocation of single APIs, the input data has to be transformed in the correct format, which can be very difficult, since sometimes the lists with alternative or coded values are not provided. For the invocation of mashups, the transformation between the inputs of one API and the outputs of the next one has to be defined. Currently, this work requires extensive manual effort and the adaption of existing Web service invocation approaches is hindered by the under-specification and the variability of the parameters.

This situation is aggravated by the fact that two thirds of the APIs do not even state the data-type of the input parameters. As a result developers need to determine the proper input format by making assumptions or through trial-and-error. In addition, the reuse of existing invocation approaches or the development of new ones is made extremely difficult, since the data-type information is simply not available. If a standard interface description language, such as WSDL, were used to describe Web APIs, not specifying the data-types would be unthinkable. However, the current state of Web APIs shows us that this in not always necessary. Since there is no common IDL, under-specification is very common and it effects in no way the level of reuse of APIs. Our data showed that there is no correlation between stating the data-type of input parameters and the number of mashups.

D. Output Formats

As it can be seen in Table IV, there are two main common output formats – XML and JSON. XML is provided in 85% of the cases and JSON in 42%, while more than one third of the APIs provide both. Further output formats include HTML, CVS, RDF, Text, object, RSS, GFF, Serialised PHP, Tab, YAML. These results show that providing support for the use of XML and JSON addresses the vast majority of the APIs.

Table IV: Output Formats

Description	Number	In %
XML	80	36
XML, JSON	53	23.9
XML and other	34	15.3
XML, JSON and other	23	10.4
only JSON	12	5.4
only other	14	6.3
JSON and other (except XML)	6	2.7
RDF	13	5.8
Total XML	190	85.6
Total JSON	94	42.4

The way of specifying how the results should be structured can be determined in two ways. Either the API provides a separate operation for every output format or it is determined though a parameter. This might present a challenge for invocation, since currently there is no commonly accepted way for stating the desired output format.

E. Invocation Details

In this section we describe our findings in relation to the invocation details commonly provided in API descriptions. The collected data is of crucial importance, since it has a direct impact on the usability of the APIs.

Table V: Invocation Details

Description	Number	In %
Provide HTTP method	134	60.4
Provide invocation URI	214	96.4

Table V shows that almost all descriptions provide the URI for invoking the API, while only about two thirds state the HTTP method to be used. This is possibly because providers assume that the method to use is GET, especially for APIs that can be invoked directly through parameterising the URI.

Table VI: Common Web API Authentication Approaches

Authentication Mechanisms	Number	In %
API Key	89	38%
No Authentication	46	19%
HTTP Basic	32	14%
Username and Password	19	8%
OAuth	14	6%
Web API Operation	12	5%
HTTP Digest	11	5%
API Key in Combination with Other Credentials	5	2%
Session Based	5	2%
Other	2	1%
Authentication Only for Data Modification	4	2%
Offer Alternative Authentication Mechanisms	16	7%

Our analysis also shows that more that 80% of the APIs require some form of authentication (Table VI). As it can be seen, using an API key (also called “developer key”, “developer token”, “token Id”, “user Id”, “user key”) is by

far the most common way of authentication (38%). It is followed by 19% of APIs, which do not require any authentication. HTTP Basic and HTTP Digest [7] are not used as often (14%, 5%), while about 6% of the APIs use OAuth [8] and 5% implement their own operations, which need to be called, before being able to invoke other operations. There are also some APIs, which require authentication only for operations, which perform data modification but require no authentication for only reading resources.

In summary, at least in 40% of the cases there is missing information required for the invocation of the APIs and 3 out of 4 APIs require some form of authentication, which means that developers would have to sign up with providers for acquiring the appropriate credentials. In addition, there is no established approach for Web API authentication but rather a landscape of different approaches. Also, about only a quarter of the APIs use a mechanism that protects the user credentials and does not transmit them directly in plain text. This shows that providers are not so much concerned with verifying the user identity and do not invest implementation work in securing the message transfer but rather prefer to employ simple measures for controlling resources usage. This is verified by the fact that less than 10% of the Web APIs use signatures and encryption.

Table VII: Way of Transmitting Credentials

Transmission Medium	Number	In %
URI	117	70%
HTTP Header	45	27%
URI or HTTP Header, Depending on the Type of Authentication and HTTP Method	6	3%

Table VII shows the most commonly used ways for transmitting authentication credentials. As it can be seen, 70% of the Web APIs send authentication information directly in the URI, while less that one third require that the HTTP header is constructed. This means that even if Web APIs require authentication, most of them do not need a custom client but can rather be invoked directly from a Web browser. These numbers are similar for invocation in general, where about one third of the APIs require the construction of the HTTP request, while the rest can be called by using the URI.

F. Completeness of the Documentation

Finally, in this section we present results for API description features, which are not strictly necessary for directly supporting service tasks such as discovery or invocation, but are useful when implementing and using the APIs. As Table VIII shows, more than 75% of the APIs provide example requests and responses. These give valuable information about the structure and the form of the request as well as of the retrieved results and, therefore, ease the development work.

We also found out that about only half of the APIs describe the used error codes. This represents a problem, since in half of the cases developers cannot determine and have no indication of what went wrong and whether the error is due to an incorrect invocation, to the connection, to missing credentials, etc.

Table VIII: Complementary Documentation

Description	Number	In %
APIs that provide an example Request	186	83.8
APIs that provide an example Response	167	75.2
APIs that describe the Error messages	118	53.1

IV. Results

As already pointed out, Web APIs face a number of challenges mainly related to the fact that currently all common service tasks such as discovery, composition and invocation require extensive manual effort. However, before any significant improvement can be achieved and suitable approaches can be devised, we need to gain a clear picture of the development process, used technologies, available information, richness of the descriptions, etc. In order to contribute directly towards this goal, in this section we derive a number of important results and conclusions, characterising the current Web API landscape.

1) Finding Web APIs on the Web requires either manual search, by using general-purpose search engines like Google and Yahoo or referring to directories like ProgrammableWeb, which are based on manual input that is sometimes inaccurate or outdated.

This result points out one of the main challenges faced by current Web API repositories. Since the API descriptions are published and updated manually by users, some of the entries are not up-to-date or no longer exist. In addition, details such as the authentication method are not always accurate. Therefore, there is a need for developing solutions for a more automated way of collecting, publishing and updating API descriptions.

2) Few APIs are highly reused, whereas most APIs, are used in very few or no mashups at all. In addition, there is no correlation between the level of reusability of APIs and their granularity.

Reusability, as indicated by the number of mashups per API, is a very important characteristic of the current Web API landscape. First, since we have no direct information about how many of the existing APIs are actually being used, the number of mashups is an indirect indication for that. Second, the frequent participation of APIs in mashups is reflected in the increased significance of certain service tasks, in this case composition, and the pieces of data required for supporting these tasks. This is made even more clear by the fact that the 222 APIs, analysed in our study, participated in a total of 1350 mashups. Therefore, future approaches

for supporting the use of APIs should especially focus on enabling the composition and creation of mashups.

3) There are three main types of Web API descriptions (RESTful, RPC-style and hybrid) but developers prefer to describe APIs in terms of operations, rather than resources.

This means that each type of Web API requires separate invocation support, which makes it even more challenging to provide support for the invocation of mashups. Currently, mashup development is based on individual solutions, which have a low level or reusability and do not contribute to the automation of a common API invocation process. The fact that most developers prefer to describe APIs in terms of operations, disregarding REST principles can be explained by looking at popular ways for defining interfaces, which are commonly based on operations and methods. Therefore developers with previous knowledge of interface description languages and a background in programming intuitively tend to formulate Web APIs in terms of operations, rather than resources that are manipulated through the HTTP methods.

4) APIs reuse in not driven by the particular type of Web API description (RESTful, RPC-style or hybrid). Therefore, the current proliferation of Web APIs cannot be attributed to the use of RESTful APIs.

We base this conclusion on the fact that the mashups percentage distribution matches almost exactly the Web API description type distribution. Our data shows no indication of RESTful APIs having a leading role in determining how APIs are described or whether they are used in mashups.

5) The description of input parameters is very flexible, allowing for the use of default values, coded values, alternative values and optional parameters. This presents a hindrance for all service tasks, especially invocation.

Service tasks that predominantly rely on the input information, such as discovery, composition and invocation, gain complexity, since the presence of some parameters is non-restrictive and the input data has to be transformed into coded or alternative values. As a result the approaches, which aim to support the use of Web APIs, should be able do deal with the flexibility of the input parameters. This is especially true for invocation, which would require the development of an integrated view on all theses diverse input forms.

6) XML and JSON are establishing themselves as the main output formats.

Even though there are no guidelines for the format of the output, currently most APIs give their results either in XML or JSON. Therefore, providing support for using and processing only these two formats, would directly contribute to the overall increase of Web API usability.

7) More than 80% of the APIs require some form of authentication.

Therefore, authentication is a vital part of the invocation process and any approach for supporting the use of APIs and mashups that disregards authentication, has very limited applicability. Currently, developers have to sign up with multiple providers in order to acquire credentials necessary for APIs participating in mashups or restrict the implementations to APIs, which are based on shared credentials such as OAuth [8].

8) Most API descriptions are characterised by under-specification.

Our data shows that two thirds of the APIs do not state the data-type of the input and 40% of the APIs do not state the HTTP method. If a standard interface description language, such as WSDL, were used to describe Web APIs, not specifying these details would be unthinkable. Since there is no common IDL, under-specification is very common and, more importantly, as our data shows it effects in no way the level of reuse of APIs.

Looking at the different results provided in this section, it becomes obvious that currently the Web API landscape is very heterogeneous and it is not possible to determine what a typical Web API description looks like. Without a doubt, all descriptions contain common pieces of information, which are required for the support of main service tasks, such as discovery, composition and invocation. However, since Web API development is not guided by standards, the diversity spreads from the structure and the form of the documentation up to the technological principles used behind the implementation. Therefore, currently the use of APIs requires extensive manual effort and the development of automated approaches is very challenging.

V. Discussion

In this section we reflect on a number of further trends and correlations that we discovered while conducting our Web API analysis. In particular, we describe how APIs from the same domain tend to have some similar features.

One interesting correlation that we detected is that APIs from the same ProgrammableWeb category tend to have the same type of description. For example, all bookmarking APIs were RPC-style, while all project management ones were RESTful. This is also true for most of the categories, where we found out that the majority of the APIs have the same type. This might be due to developers investigating competing providers and their services and, therefore, being influenced by the way APIs with similar functionalities are structured and described. Also for some use cases it is more intuitive to base the description on resources, while for others using operations is more natural. For example, getting a set of news articles or information about an artist can easily be described based on resources (GET http://example.com/News and GET http://example.com/Artist?name=madonna), while

determining a route between two locations or retrieving the temperature in a city can better be realised with operations.

In addition to having similar types of descriptions, we discovered that APIs from the same category usually have similar authentication mechanisms. For example, most governmental and medical APIs require no authentication, while job search and general search APIs commonly use an API key. This again can be attributed to developers comparing their API with other APIs with similar functionality and as a result adapting similar authentication measures. However, certain domains should naturally be very accessible, while others related to more private or confidential information should be supported by stronger authentication measures.

The survey also provided some important information about the Web API description forms. In particular, none of the analysed APIs used WSDL [6] or WADL [5] and the majority of the APIs are documented directly in HTML Web pages. A search for WADL documents in Google returns only around 160 matches, but certainly not all of these represent actual APIs (search was done on 19.05.2010). This number should be compared to over 2000 APIs currently registered in the ProgrammableWeb directory. In addition, some of the descriptions were in PDF, which requires downloading the documentation and makes crawling for APIs and automated processing more difficult.

VI. Related Work

Up-to-date the current state of Web APIs, including different description forms, input types, invocation details, etc., has remain unexplored. However, there are two similar studies, devoted to investigating Web services on the Web. The authors in [10] provide a study on Web services, focusing on deriving statistics based on operations analysis, size analysis, words distribution and function diversity analysis by using the Google API. This study is based only on a few Web service characteristic and is restricted to only one source. A broader and more complete study is given by [9]. The authors have developed a crawler for collecting metadata about service interfaces available through repositories, portals and search engines. The gathered data is used to determine statistics about object sizes, type of technology and functioning of the Web services, among others. In comparison to previous studies, this one also provides conclusions about the status of Web services and what percentage of the Web services are considered to be active and responsive.

VII. Conclusion and Future Work

Currently, finding, interpreting and invoking Web APIs requires extensive human involvement due to the lack of API machine-processable descriptions. However, before any significant progress and improvement can be made to the existing practices and technologies for Web APIs, we need to reach a deeper understanding of how APIs are developed and exposed, what kind of descriptions are available, how they are represented and how rich these descriptions are. In this paper, we contribute directly to this goal by providing a thorough analysis of the current state of Web APIs based on investigating six groups of main characteristic features including – general information, type of Web API, input parameters, output formats, invocation details and complementary documentation. By using the collected data, we can better realise what the current difficulties are, which problems need to be addressed, and how should supporting mechanisms be devised. In this sense, we show that RESTful services are not the driving force behind the current Web API proliferation and that Web API descriptions are characterised by under-specification, where important information such as the data-type and the HTTP method are commonly missing.

Future work will involve the conduction of the study, over the same set of Web APIs, in one year. In this way we will have an updated view of the Web API landscape and can make statements about the changes and developments. In addition, we are planning on investigating some further correlations such as the ones between the domain and the lever of reuse, or the granularity and the domain.

References

[1] L. Richardson, S. Ruby: RESTful Web Services. O'Reilly Media, May 2007.

[2] R. T. Fielding: Architectural styles and the design of network-based software architectures. PhD thesis, University of California, 2000.

[3] C. Pautasso: RESTful Web service composition with BPEL for REST. Data and Knowledge Engineering journal, 68:851-866, 2009.

[4] K. Gomadam, A. Ranabahu, M. Nagarajan, A. Sheth, K. Verma: A Faceted Classification Based Approach to Search and Rank Web APIs. In proceedings of the 2008 IEEE International Conference on Web Services, 2008.

[5] M. J. Hadley: Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006. Available at https://wadl.dev.java.net.

[6] Web Services Description Language (WSDL) Version 2.0. W3C Recommendation, June 2007. Available at http://www.w3.org/TR/wsdl20/.

[7] J. Franks, P. Hallam-Baker, J. Hostetler: HTTP Authentication: Basic and Digest Access Authentication RFC2617. The Internet Society, 1999.

[8] M. Atwood, et al: OAuth Core 1.0 Specification. http://oauth.net/core/1.0/.

[9] E. Al-Masri, M. H. Qusay: Investigating web services on the world wide web. In Proceedings of WWW, pp.795-804, 2008.

[10] Y. Li, Y. Liu, L. Zhang, G. Li, B. Xie, and J. Sun: An Exploratory Study of Web Services on the Internet. In Proceedings of ICWS, pp. 380-387, 2007.