# Using semantics for automating the authentication of Web APIs

## Conference or Workshop Item

# Using Semantics for Automating the Authentication of Web APIs

Maria Maleshkova[1], Carlos Pedrinaci[1], John Domingue[1], Guillermo Alvaro[2], Ivan Martinez[2]

[1] Knowledge Media Institute (KMi)
The Open University, Milton Keynes, United Kingdom
{m.maleshkova, c.pedrinaci, j.b.domingue}@open.ac.uk
[2] Intelligent Software Components (iSOCO). Madrid, Spain
{galvaro, imartinez}@isoco.com

**Abstract.** Recent technology developments in the area of services on the Web are marked by the proliferation of Web applications and APIs. The implementation and evolution of applications based on Web APIs is, however, hampered by the lack of automation that can be achieved with current technologies. Research on semantic Web services is therefore trying to adapt the principles and technologies that were devised for traditional Web services, to deal with this new kind of services. In this paper we show that currently more than 80% of the Web APIs require some form of authentication. Therefore authentication plays a major role for Web API invocation and should not be neglected in the context of mashups and composite data applications. We present a thorough analysis carried out over a body of publicly available APIs that determines the most commonly used authentication approaches. In the light of these results, we propose an ontology for the semantic annotation of Web API authentication information and demonstrate how it can be used to create semantic Web API descriptions. We evaluate the applicability of our approach by providing a prototypical implementation, which uses authentication annotations as the basis for automated service invocation.

## 1 Introduction

Web services provide means for the development of open distributed systems, based on decoupled components, by overcoming heterogeneity and enabling the publishing and consuming of functionalities of existing pieces of software. Recently the world around services on the Web, thus far limited to "classical" Web services based on SOAP and WSDL, has been enriched by the proliferation of Web applications and APIs, also referred to as RESTful services [1], when conforming to the REST architectural principles. Web APIs are characterised by their relative simplicity and their natural suitability for the Web, relying directly on the use of URIs, for both resource identification and interaction, and HTTP for message transmission. Many popular Web 2.0 applications like Facebook, Google, Flickr and Twitter offer easy-to-use, publicly available APIs, which not

only provide simple access to different resources but also enable combining heterogeneous data coming from diverse services, in order to create data-oriented service compositions called mashups.

Despite their success, Web APIs are currently facing a number of limitations. While the development, publication and use of Web services is guided by standards and specifications, the Web API landscape is much more heterogeneous and diverse. This heterogeneity is especially present in the forms and structure of the documentation, since currently most Web API descriptions are given directly in text/HTML as part of a webpage. Providers publish the APIs in a way that they see fit, following no particular guidelines and conforming to no particular standards. As a consequence, in order to use Web APIs, developers are obliged to manually locate, retrieve and interpret heterogeneous documentation, and subsequently develop custom tailored software, which has a very low level of reusability. The diversity of the Web APIs is accompanied by a wide range of used authentication approaches, which hinder the automated Web API invocation. The lack of a common structured language for describing Web APIs is addressed by some initial proposals [2], [3], while lightweight annotations over Web API descriptions [4], [5] have been developed as means for overcoming the existing heterogeneity and providing basic support for service task automation. Still, up to date, the importance of authentication as part of the invocation process has been neglected. As our study points out (see Section 4.2) the majority of the Web APIs require some form of authentication but none of the existing formalisms and annotation approaches deal with this. Moreover, none of the available tools, which provide developer support for creating mashups, such as Yahoo Pipes and DERI Pipes[3], handle authentication in an integrated way and it still needs to be addressed separately. As a result, the invocation of individual Web APIs and their use within mashups, requires extensive manual development work, independently of whether they are semantically annotated or not.

In order to support the automated invocation of Web APIs, we propose the use of semantic annotations over the existing heterogeneous HTML descriptions. As shown by our study, the invocation of Web APIs requires authentication in more than 80% of the cases, but currently there is no description formalism or semantic annotation approach, which addresses this, and commonly the need for authentication support is simply neglected. Therefore, we provide an ontology for the annotation of authentication information on top of Web API descriptions, which covers all authentication mechanisms identified by our study and is extendable to cover further ones. We show how semantic descriptions of authentication details can be created and validate our approach by providing a prototype of an invocation system, which effectively uses the created annotations to support the automated invocation of Web APIs.

The remainder of this paper is structured as follows: Section 2 provides a motivating example that illustrates the challenges related to Web API authentication, while Section 3 gives an overview of previous work in the area of semantic Web API descriptions and Web service security. A detailed analysis of current

[3] `http://pipes.yahoo.com/pipes/`, `http://pipes.deri.org/`

API authentication approaches is given in Section 4. Based on the Web API survey, in Section 5 we propose an ontology and an implementation for supporting the automated authentication, by using lightweight semantic annotations. Section 6 presents an overview of related work and Section 7 concludes the paper.

## 2 Motivating Example

In this section we present a simple example, which demonstrates the necessity of authentication information during the invocation of Web APIs, and use it throughout the paper to illustrate the here proposed annotation approach. In particular, we describe one of the operations of the Last.fm Web API[4]. The Last.fm API enables developers to access and use the Last.fm data. This popular website for music claims more than 40 million active users and provides details about artists, albums, events and user-specific information, such as playlists.

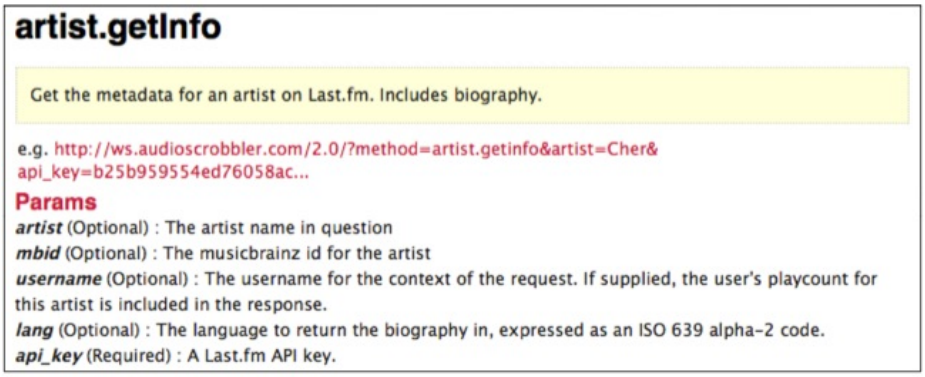


**Fig. 1.** Extract from the Last.fm API

Figure 1 shows the Web API operation for getting the details for a particular artist. The provided data can be used directly or as part of a mashup, where artists details are combined with latest charts news, for example. However, since the Web API is described solely in HTML, the discovery and interpreting of the documentation have to be done manually. Moreover, even if the Last.fm API were semantically annotated or had a machine-processable description, for example in WADL, the automated invocation would still not be possible because of the necessity of providing an authentication API key, which cannot be captured with existing description forms. The work presented in this paper is targeted at addressing precisely this problem.

[4] `http://www.last.fm/api`

## 3 Background

Since the advent of Web service technologies, research on semantic Web services (SWS) has been devoted to reduce the extensive manual effort required for manipulating Web services. The main idea behind this research is that tasks such as discovery, negotiation, composition and invocation can have a higher level of automation, when services are enhanced with semantic descriptions of their properties. Similarly to "classical" Web services, Web API-related tasks also require a lot of developer involvement and face even further difficulties, since there is no established common formalism for describing Web APIs. In order to address this, lightweight annotations over API descriptions have been proposed as means for achieving a higher-level of automation.

Currently, there are two main contributions aiming at using semantics to support the automation of common Web API service-related tasks. Both approaches rely on marking service properties within the HTML description and subsequently liking these to semantic entities. MicroWSMO [4] is a formalism for the semantic description of Web APIs, which is based on adapting the SAWSDL [6] approach for enhancing service properties with semantic information. MicroWSMO uses microformats for adding semantic information on top of HTML service documentation, by relying on hRESTS [7] for marking service properties. Listing 1.1 shows the hRESTS annotation of the Last.fm API, where the different service properties are identified via HTML tags.

**Listing 1.1.** Last.fm Web API

```
<div class="service" id="service1"><h1>Last.fm Web Services</h1>
<div class="operation" id="op1"><h2><span class="label">artist.getInfo</span></h2>
<div>Get the metadata for an artist on Last.fm. Includes biography.</div>
<span class="address">http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&amp;
 artist =Cher&amp;amp;api_key=xxx</span>
<div class="input" id="input1">
 <span>artist</span> (Optional) : The artist name in question<br>
 <span>lang</span> (Optional) : The language to return the biography in.<br>
 <span>api_key</span> (Required) : A Last.fm API key.<br></div>
<div class="output" id="output1">Artist</div></div></div>
```

Another formalism is SA-REST [5], which also applies the grounding principles of SAWSDL but instead of using hRESTS relies on RDFa [8] for marking service properties. Similarly to MicroWSMO, SA-REST enables the annotation of existing HTML service descriptions by identifying service elements and linking these to semantic entities. The main differences between the two approaches are not the underlying principles but rather the implementation techniques.

Both MicroWSMO and SA-REST, provide a solid foundation for the use of semantics as the basis for automating common service tasks. However, they are very lightweight and the automation support that they provide is limited. More importantly, all existing approaches neglect the need for addressing the automation of Web API authentication. Therefore, we use existing models for the semantic description of Web APIs as the basis for an incremental approach for reflecting authentication information.

### 3.1 WS-Security

The issues of authentication and security have already been tackled in the context of WSDL and SOAP-based Web services. The result is a unified Web service security standard. WS-Security [9] specifies a set of feature extensions to SOAP messaging, in order to provide message integrity and confidentiality. In addition, it also provides a mechanism for associating security tokens with message content and allows for a variety of signature formats and encryption algorithms. As a result, the defined enhancements provide support for ensuring that the sent message is not altered by a third party (message integrity), that its content cannot be read by anyone but the designated client or server (confidentiality) and that the user has the necessary credentials in order to access particular resources (authentication).

WS-Security addresses the main security issues in the context of Web services. However, in contrast to WSDL-based services, Web APIs are proliferating autonomously without the creation of standards and independently from Web services. The result is a very heterogeneous world of Web APIs, where security issues such as confidentiality and message integrity, guaranteed by the WS-Security standard, are not considered as crucial. In fact, as our study shows, security in the context of Web APIs is reduced only to authentication, which in turn serves mainly the purposes of access control, where providers rather want to restrict and track the number of requests, instead of providing data integrity.

## 4 Investigating Authentication for Web APIs

In order to become aware of currently used Web API authentication approaches, we conducted a study, analysing 222 Web APIs from the ProgrammableWeb[5] directory. ProgrammableWeb is a popular API directory, providing information about 2002 APIs and 4827 mashups (visited June 2010). 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 analyzed Web APIs from each category were randomly chosen, however, since some categories have only one or two entries, the analyzed number of Web APIs per category varies. As a result the survey covered 18% of the REST ProgrammableWeb APIs (1235 APIs at the time of the study, February 2010). Therefore, we consider the following results to be representative for the directory and in general, since ProgrammableWeb is currently the biggest directory[6].

In the following sections we first provide an overview of common authentication approaches, as identified by our Web API analysis, and then layout the results and conclusions of our Web API survey.

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

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

### 4.1 Common Authentication Approaches

Currently, as our survey shows, most Web APIs use one of five authentication mechanisms. We differentiate between approaches based only on authentication credentials (API key or username and password), approaches using a transmission security protocol (HTTP Basic Authentication, HTTP Digest Authentication and OAuth), and approaches that use different parts of the HTTP request in order to transmit the authentication information. We start by describing authentication mechanisms relying only on the input credentials.

**API Key.** Currently, the most common way of Web API authentication is via API key (also called "developer key", "developer token", "token Id", "user Id", "user key"). Web APIs using this mechanism include Last.fm (`http://www.last.fm/api`) and Remember the Milk (`http://www.rememberthemilk.com/services/api/`). This authentication mechanism does not have any security measures for the message integrity and confidentiality but is rather only based on the necessary credentials. The user only needs to provide the API key, which is received by signing up for the particular Web API. The key is transmitted either as a parameter in the Web API URI or directly in the HTTP request. Each client that provides a valid API key is permitted access to the requested resources. This approach is very simple to use and to implement. However, since the API key is not protected in any way during the message transmission, but is rather sent directly as plain text, this method is suitable for Web API providers, who only want to somehow restrict the access to the available resources.

**Username and Password.** Similarly, to authentication via API key, authentication via username and password is also only based on the required credentials. It provides no message encryption or signature and the login details are transmitted as parameters of the request URI or are included in the HTTP request. Example Web APIs include Happenr (`http://www.happenr.com/webservices/`, for example `http://happenr.com/webservices/getEvents.php?username=xxx&password=xxx&town=London`) and FileSocial (`http://filesocial.com/api/docs`). The user only needs to create an account for the particular Web API and can use the username and password (in some cases email and password, telephone number and pin, username and token or API key and private key) to access resources. Similarly to the authentication via API key, this approach is only suitable for providers who want to restrict the traffic and the number of requests to their APIs.

The first two authentication mechanisms are only based on the required credentials, while the following approaches, including HTTP Basic Authentication and HTTP Digest Authentication, are transmission security protocols. These, provide a higher level of security for the login details and the client's message.

**HTTP Basic Authentication** [10] provides a simple way for user authentication. It is based on a challenge-response model, where the HTTP server requests and validates the authentication of the Web client. Example Web APIs include Assembla (`https://www.assembla.com/wiki/show/breakoutdocs/Assembla_REST_API`) and Basecamp (`http://developer.37signals.com/basecamp/`). In order to access a Web API operation, which requires authenti-

cation, the client needs to provide the corresponding username and password in the form of an authentication header (with value Base64encode [11] of the string `username+":"+password`). The Base64-encoded string is transmitted and decoded by the receiver, resulting in the colon-separated user name and password strings, which are checked against the expected values.

This type of authentication is very simple and is supported by all popular Web browsers. However, although it uses Base64 encoding, it does no encryption and the username and password can directly be decoded from the transmitted message. Therefore, this type of authentication is only suitable for Web APIs with lower data security demands.

**HTTP Digest Authentication** [10] follows the same process as the HTTP Basic one – request, credentials challenge and response. However, it only transmits a digest of the username and password, which cannot be directly decoded. Example Web APIs include Talis (`http://n2.talis.com/wiki/Platform_API`) and AdSpeed (`http://www.adspeed.com/Knowledges/830/AdSpeed_API/AdSpeed_API_Overview.html`, for example `http://api.adspeed.com/?method=METHODNAME&param1=VALUE&param2=VALUE&md5=`**SIGNATURE**). The first time a client sends a request to the server, the server responds with a nonce (a random string) and the realm (typically a description of the computer or system being accessed). The client uses the username and password, to compute the digest response (result of `MD5(username:realm:password)`) and the digest of the nonce (usually by using MD5 [12]), which are put in the response. The server processes the response by retrieving the stored password for the user and testing the nonce. If the nonce is correct, the response digest is checked by using the nonce, username and password to compute a digest and compare it to the received one. If the two digests match, the client is allowed access to the resources.

**OAuth** [13] is a protocol for making authenticated HTTP requests by using a token. It enables users to share private resources stored on one website with another one by using a token, which is an identifier denoting an access grant with specific scope, duration, and other attributes, instead of the username and password. Therefore, OAuth supports the interoperability and the combining of resources coming from different websites, in a way that is transparent for the user. Example Web APIs include Fire Eagle (`http://fireeagle.yahoo.net/developer/documentation`) and Delicious (`http://delicious.com/help/api`).

Whenever a Web API (or a website) needs to access resources from another Web API, the user is asked to provide his/her access information for the host Web API, while in the background, OAuth creates a token, which can be used by other APIs to gain access to the resources. As a result the username and password are kept private and unavailable for third-party websites and APIs, while the interoperability is still facilitated. This authentication approach is extremely important in the context of mashups, since it does not require that the user provides credentials for every Web API included in the composition, but rather relies on token-based user-transparent handling of authentication.

So far we have described authentication based on different credentials and on using different authentication mechanisms. In addition, there are also two

main ways of transmitting the authentication information. One very common way is to directly provide the API key or username and password as parameters in the request URI. For example Last.fm (`http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&artist=Cher&apikey=XXX`) and Fire Eagle (`https://fireeagle.yahooapis.com/api/0.1/`) use this approach. Since the authentication credentials are not protected in any way, this way of sending data is suitable for openly available resources, where providers want to restrict the access to the API but are not concerned with enforcing access rights. The other common way of sending authentication credentials is directly in the HTTP request. This method is somewhat more complex because the client needs to construct the request, instead of only calling a parameterized URI. However, it enables a higher level of security, since the information can be encrypted and signed. For example, this way of sending information is commonly used by the HTTP digest authentication approach.

In summary, current authentication approaches have three main characteristics: 1) the required credentials, 2) the used authentication protocol, and 3) the way of sending the authentication information.

### 4.2 Web API Survey Results

After introducing the most common authentication approaches, in this section we focus on describing the results and the main findings of our Web API study. Table 1 shows the results of our analysis in terms of Web API authentication approaches. As it can be seen, using an API 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 are not used as often (14%, 5%), while about 6% of the APIs use OAuth 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

**Table 1.** 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% |

data modification but require no authentication for only reading resources. The sum of all APIs is greater than 222 because, APIs that offer more that one authentication mechanism were counted more than once.

It is important to point out that currently 81% of the Web APIs require some form of authentication. Therefore, providing support for the annotation and automation of authentication is crucial for Web API use. 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 access control. This is verified by the fact that less than 10% of the Web APIs use signatures and encryption.

**Table 2.** 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 2 shows the most commonly used ways of transmitting authentication credentials. As it can be seen, 70% of the Web APIs send authentication information directly in the URI, while less than 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.

The survey also delivered some important information about the Web API description forms. In particular, none of the analyzed APIs used WSDL [3] or WADL [2] and the majority of the APIs are documented directly in HTML Web pages. The main conclusions of the Web API survey can be summarised as follows:

1. More that 80% of the Web APIs require authentication. Therefore authentication is a vital part of the invocation process and any approach disregarding authentication information has very limited support.
2. The currently used authentication approaches are very heterogeneous and there is no commonly accepted way for addressing Web API authentication.
3. Only about 25% of the Web APIs use an authentication approach that protects the user credentials and/or the content of the message.

## 5 Supporting the Automation of Web API Authentication

As highlighted by the Web API survey, authentication is essential and any general purpose solution aiming at supporting the invocation of Web APIs or mashups would necessarily be restricted to only 20% of the APIs or requires customisation. In addition, the authentication approaches are very diverse and similarly to Web API descriptions in general, authentication details are not described in a machine processable way but are given as part of documentation websites. In order to overcome the heterogeneity and provide means for the automatic recognition and processing of authentication details, we propose that Web API descriptions are annotated with semantic information about authentication. We next present the main design principles followed when deriving the proposed authentication ontology and continue by describing it in detail.

### 5.1 Design Principles

Guided by the results of the Web API study, we analyzed the collected data and derived an authentication ontology, which enables the annotation of authentication information as part of a semantic Web API description. The process of defining this ontology was guided by a number of competency questions and design principles. First, we started by identifying the cases, in which authentication is required, and the information that is needed. Relevant information in this respect is: *"Does the service require authentication?"*, *"Which operations require authentication?"*, *"What kind of authentication is used?"*, and *"What is the required information to complete the authentication?"*. As we concluded, based on the analysis of common authentication approaches, authentication has three main characteristic including the required credentials, the used authentication protocol, and the way of sending the authentication information. Therefore, we can identify the information necessary for supporting a particular authentication mechanism by determining: *"What are the required credentials?"*, *"What is the used authentication protocol?"* and *"How is the authentication information transmitted?"*. In addition to the competency questions, used for identifying the information that needs to be captured by the authentication ontology, we implemented some complementary requirements, which are specified in the form of design principles. The principles are as follows:

1. The ontology should cover all common authentication approaches identified by the Web API study.
2. The ontology should be extendable to cover further mechanisms.
3. The ontology should capture the information required for the automation of authentication as part of the invocation process.
4. The ontology should be compatible with existing semantic annotation approaches, such as MicroWSMO and SA-REST.
5. The ontology should support simplicity of use for making annotations.
6. The ontology should aim to be minimal but capture the necessary information for supporting the authentication.

The so designed ontology is not bound to any particular annotation formalism, but can be used as an extension by simply attaching it to the service and operation elements. In the following section we introduce the authentication ontology and show how it supports the automation of the invocation of Web APIs by using it as part of an authentication engine.

### 5.2 Authentication Ontology

Figure 2 depicts the Web API authentication ontology with namespace *waa* (Web API Authentication), which consists of three main classes–*AuthenticationMechanism*, *Credentials* and *TransmissionMedium*[7].

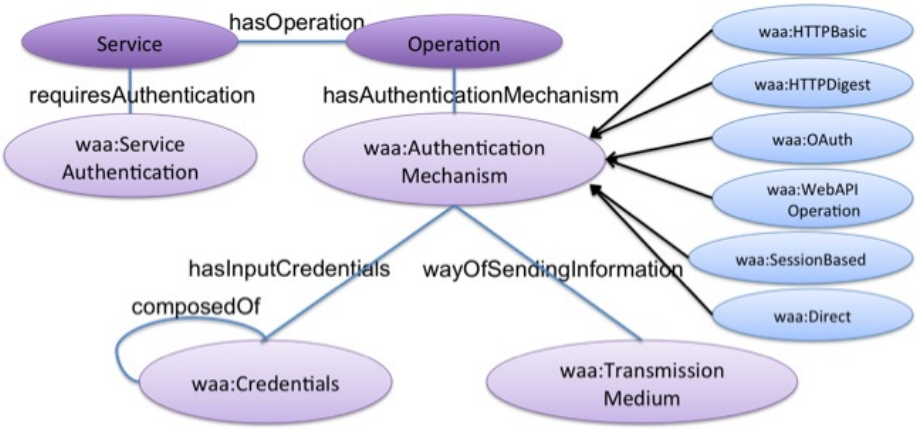


**Fig. 2.** Web API Authentication Ontology

The *AuthenticationMechanism* class has six subclasses, corresponding to common authentication mechanisms, where the *Direct* subclass is used to describe approaches, which rely on using only credential details and employ no authentication protocol. The *Credentials* class has a number of instances including *API_Key*, *Username*, *Password* and *OAuth Credentials*, which can be combined to produce composite credentials, such as authentication through username and password. The *composedOf* relationship as well as the class *AuthenticationMechanism*, which can have further subclasses, represent points of extensibility for the ontology. The *Service* class has a relationship to the *ServiceAuthentication* class, which has three instances including *All*, *Some* and *None* that are used to point out that the service requires authentication for all its operations, for only some of them or for none of them. The *TransmissionMedium* has two instances (*ViaHTTPHeader* and *ViaURI*), used to describe that the credentials are sent by using only the URI or through constructing an HTTP header.

[7] The ontology is available at `http://purl.oclc.org/NET/WebApiAuthentication`

The *Service* and *Operation* classes lack a namespace, because they serve as placeholders that can be replaced by the service and operation elements of any Web API model whether it is semantic, such as MicroWSMO or SA-REST, or not. In this way, the ontology can be used as an extension to existing formalisms or independently from them.

In order to show how the authentication ontology can be used to annotate Web APIs, we have taken the Last.fm motivating example from Section 2 and provide its HTML annotation (Listing 1.2) and semantic description (Listing 1.4). We apply the annotation approach presented in [14], where Web API descriptions are enhanced with semantic information by using MicroWSMO and SWEET as a supporting tool. In the example we use the *wsl* (`http://www.wsmo.org/ns/wsmo-lite`) namespace for WSMO-Lite, which is used in MicroWSMO annotations for the service model. However, as pointed out, the authentication annotations can be assigned to any operation and service model elements.

**Listing 1.2.** Example MicroWSMO Authentication Annotation

```
1  <div class="service" id="service1"><h1>Last.fm Web Services</h1>
2  <a rel="model" href="http://purl.oclc.org/NET/WebApiAuthentication#All">
3  <div class="operation" id="op1"><h2><span class="label">artist.getInfo</span></h2>
4  <a rel="model" href="http://purl.oclc.org/NET/WebApiAuthentication/LastFm">
5  <span class="address">http://ws.audioscrobbler.com/2.0/?method=artist.getinfo...</span>
6  <div class="input" id="input1">...</div>
7  <div class="output" id="output1">Artist</div></div></div>
```

Listing 1.2 shows the annotated HTML of the Last.fm API. The Web API requires authentication for all its operations (Line 2) and has authentication information for the *artist.getInfo* operation reflected in Line 4. The model reference contains a URI pointing to a particular instance of the *AuthenticationMechanism* class, which contains details about the operation requiring an API key, which is sent in the URI without the use of any authentication protocols.

**Listing 1.3.** Example Instance of the AuthenticationMechanism Class

```
1  @prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix waa:    <http://purl.oclc.org/NET/WebApiAuthentication#> .
3  <http://purl.oclc.org/NET/WebApiAuthentication/LastFm> rdf:type waa:Direct ;
4    waa:hasInputCredentials waa:API_Key ;
5    waa:wayOfSendingInformation waa:ViaURI .
```

Listing 1.3 shows how this instance of the *AuthenticationMechanism* class looks like. As it can be seen, the capturing of authentication information with the provided Web API authentication ontology is very simple and easy to apply.

**Listing 1.4.** Example RDF Authentication Annotation

```
1  :service1 rdf:type wsl:Service ;
2       rdfs:isDefinedBy <http://www.last.fm/api/show?service=267> ;
3       waa:requiresAuthentication waa:All .
4  :operation1 rdf:type wsl:Operation ;
5       rdfs:label " artist.getInfo" ;
6       hr:hasAddress "http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&..." ;
7       waa:hasAuthenticationMechanism <http://purl.oclc.org/NET/WebApiAuthentication/LastFm> .
8  <http://purl.oclc.org/NET/WebApiAuthentication/LastFm> rdf:type waa:Direct ;
9       waa:hasInputCredentials waa:API_Key ;
10      waa:wayOfSendingInformation waa:ViaURI .
11 :service1 wsl:hasOperation :operation1 .
```

Based on the annotated HTML, the authentication information can easily be extracted in RDF (Listing 1.4) by using a simple XML transformation. All examples are available at `http://sweet.kmi.open.ac.uk/examples/`.

### 5.3 Authentication Engine

In this section we show how the authentication Web API annotations can be used to support the automated invocation of services. The contribution described here is implemented as part of SPICES[8] [15] (Semantic Platform for the Interaction and Consumption of Enriched Services), a platform for the easy consumption of services based on their semantic descriptions. In particular, SPICES supports both the end-user interaction with services and the invocation process itself, via the generation of appropriate user interfaces. Typically, the user is presented with a set of fields, which must be completed to allow the service to execute, and these fields cove input parameters as well as authentication credentials.

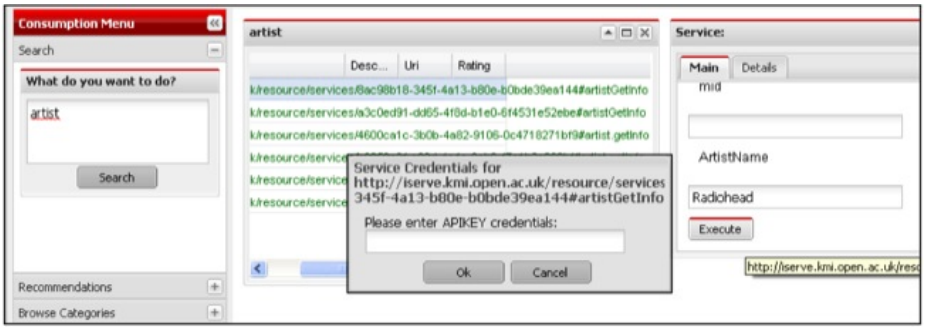


**Fig. 3.** Invoking the Last.fm API

Dealing with the different types of credentials and the way they have to be used, is the purpose of an Authentication Engine, part of SPICES, developed as a REST service, which is capable of handling the storage and retrieval of credentials for different Web APIs. This engine has the necessary logic to support the user in his/her interaction with services. In particular, if the engine has the credentials for a given service, thanks to the authentication annotations described previously, it is able to create a suitable request including the credentials. If the authentication credentials are not available yet, based on the authentication annotations, the authentication engine will be aware of the missing credentials and will prompt the user to provide them. Currently, the authentication engine plays the role of a trusted party, since it accesses and stores all user credentials. However, SPICES is only a prototypical implementation, with the main focus on supporting the invocation of Web APIs, and the authentication engine represents

[8] `http://soa4all.isoco.net/spices`

an initial practical application of the here presented approach. Therefore some issues such as appropriately storing and managing user credentials, still need to be addressed. However, since more than 70% of the authentication approaches do not protect the user credentials, this issue is not so crucial.

Figure 3 shows how the authentication engine prompts the user for the Last.fm API key, based on the API annotation, during the process of invoking the *artist.getInfo* operation. In this way the authentication engine can collect the required credentials and compose an API request, which together with the provided input information, supports the automated Web API invocation.

## 6 Related Work

In this section we describe further existing Web API authentication approaches, which address different challenges in the context of authentication but have not yet reached greater popularity.

**Web-key** [16] is an authentication mechanism, especially designed to tackle the difficulties arising in the context of mashup authentication. Web-key is an https URL convention for representing a transferable permission in a Web application. It binds each permission issued from the Web application to a randomly generated bit string (key), which is transmitted in the fragment segment of the URL (for example https://www.emaple.com/app/**#mhbqcmmva5ja3**). The keys are generated on behalf of the user for every Web API that is part of the composition. In this way, each Web API has its unique key, instead of the user having to share his username and password across all composite APIs. However, this approach is fairly new and its adoption is still limited.

Another authentication mechanism is **FOAF+SSL** [17]. FOAF+SSL is a simple protocol for RESTful authentication, which enables a one-click signing into websites by using a browser as the client application. It requires the user to enter neither a password nor an identifier but rather uses SSL and a custom trust protocol. The custom trust protocol is based on authentication certificates, which contain semantic descriptions of the authentication information in the subject alternative Name URI. FOAF+SSL presents a novel authentication approach, which includes Semantic Web principles in the authentication process. It remains to be seen how well it will be adopted for Web API implementations.

**OpenID**[9] targets to solve the problem of one user being forced to have many different Web application and API accounts, in order to be able to execute a mashup. It is a method based on using a single login at a trusted provider to automatically gain access to other websites. In this way the user can log into different services with the same digital identity, where these services trust the authentication body. Website providers, which use OpenID include AOL, IBM, Microsoft and others. OpenID is often seen as a complimentary approach to OAuth, where OpenID credentials can be used for generating OAuth tokens.

[9] http://openid.net

**XAuth**[10] provides an approach for extending authenticated user services throughout the Web by issuing user browser tokens for each of the participating services. In this way the provider can recognize, which users are logged into the services and not only give access to resources but also give additional relevant options. A different approach is followed by **Yadis**[11], who instead of suggesting a new authentication mechanism, propose means for automatically detecting, which authentication protocol a particular system is most likely to use. Therefore, Yadis addresses the question of how do we know, what authentication needs to be used, by providing a service discovery system that determines automatically, without end-user intervention, the most appropriate protocol to use.

The approach, which we propose in this paper, differs from Web-key, FOAF+SSL, XAuth and Yadis and other authentication mechanisms because we are not suggesting to alter the current Web API authentication landscape by introducing a common standard. Instead, based on a study of current Web API authentication mechanisms, we provide a lightweight model and an approach for the annotation of APIs. The resulting semantic descriptions serve as the basis for automating the Web API invocation process.

In addition to the here listed authentication mechanisms, there is also one approach that uses semantic Web service descriptions in order to capture authorisation and privacy service properties [18]. The authors suggest that privacy and authentication policies should be incorporated into the OWL-S Web service descriptions. This additional information can then be integrated into the service matchmaking process. Similarly to our approach, this approach uses semantic descriptions for capturing authentication information. However, it is suitable only for WSDL-based services annotated in OWL-S.

## 7 Conclusion and Future Work

Nowadays, finding, interpreting and invoking Web APIs requires extensive human involvement due to the lack of API machine-processable descriptions. Efforts like SA-REST and MicroWSMO aim to overcome this difficulty and provide basic support for the automation of common service tasks such as discovery and composition. However, currently none of the existing approaches support the automated authentication as part of the Web API invocation process. As a result, developers are required to manually retrieve and interpret the HTML documentation, to signup with API providers, in order to receive access credentials, and to implement support for the different authentication protocols. In addition, none of the existing frameworks for supporting the creation of mashups, such as Yahoo Pipes and DERI Pipes, enable the handling of authentication in an integrated way and it has to be addressed with additional manual effort.

Our Web API study shows that more than 80% of the APIs require authentication, which makes authentication a vital part of the invocation process and any invocation approach disregarding authentication information has very

[10] http://xauth.org
[11] http://yadis.org

limited support. Therefore, we propose the annotation of authentication information by using an authentication ontology, which overcomes Web API heterogeneity and provides the basis for automated authentication handling. We base the annotation approach on a thorough study of current Web API authentication mechanisms and show how it can be used as input to SPICES and the authentication engine, in order to support the automated invocation of Web APIs. Future work will focus on further developing the authentication engine.

**Acknowledgments** The work presented in this paper is partially supported by EU funding under the project SOA4All (FP7 - 215219).

## References

1. L. Richardson, S. Ruby: RESTful Web Services. O'Reilly Media, May 2007.
2. M. J. Hadley: Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006. Available at `https://wadl.dev.java.net`.
3. Web Services Description Language (WSDL) Version 2.0. W3C Recommendation, June 2007. Available at `http://www.w3.org/TR/wsdl20/`.
4. J. Kopecký, T. Vitvar, D. Fensel, K. Gomadam: hRESTS & MicroWSMO. Technical report. Available at `http://cms-wg.sti2.org/TR/d12/`, 2009.
5. A. P. Sheth, K. Gomadam, J. Lathem: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. IEEE Internet Computing, 11(6):91-94, 2007.
6. J. Kopecký, T. Vitvar, C. Bournez, J. Farrel: SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing, 11(6):60-67, 2007.
7. J. Kopecký , K. Gomadam, T.Vitvar: hRESTS: an HTML Microformat for Describing RESTful Web Services. In Proceedings of International Conference on Web Intelligence (WI-08), 2008.
8. RDFa in XHTML: Syntax and Processing. Proposed Recommendation, W3C, September 2008. Available at `http://www.w3.org/TR/rdfa-syntax/`.
9. A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker: Web Services Security: SOAP Message Security 1.1. (WS-Security 2004), 2006.
10. J. Franks, P. Hallam-Baker, J. Hostetler: HTTP Authentication: Basic and Digest Access Authentication RFC2617. The Internet Society, 1999.
11. N. Freed, N. Borenstein: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. `http://tools.ietf.org/html/rfc2045`.
12. The MD5 Message-Digest Algorithm: `http://tools.ietf.org/html/rfc1321`, Visited 6/2010.
13. M. Atwood, et al: OAuth Core 1.0 Specification. `http://oauth.net/core/1.0/`.
14. M. Maleshkova, C. Pedrinaci, J. Domingue: Supporting the creation of semantic RESTful service descriptions. In Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2) at 8th International Semantic Web Conference, 2009.
15. G.Álvaro, I. Martínez, J.M. Gómez, F. Lecue, C. Pedrinaci, M. Villa, G. Di Matteo: Using SPICES for a Better Service Consumption. Poster at the 7th Extended Semantic Web Conference (ESWC), 2010.
16. T. Close: Web-key: Mashing with Permission. In Proceedings of Web 2.0 Security and Privacy, 2008.
17. H. Story, B. Harbulot, I. Jacobi, M. Jones: FOAF+SSL: RESTful Authentication for the Social Web. In SPOT2009 European Semantic Web Conference, 2009.
18. L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, K. Sycara: Authorization and Privacy for Semantic Web Services. IEEE Intelligent Systems 19, 4 (Jul. 2004).