

Steffen Stadtmüller, Jorge Cardoso, and Martin Junghans

Summary

The chapter looks at how to enrich the description of cloud services with semantic knowledge. This enrichment is conducted using Linked USDL (Unified Service Description Language), a service description language built with semantic web technologies. Linked USDL provides a business and technical envelope to describe services' general information and their Web API. This improves the search and contracting of services over the web. Using the LastFM cloud service as a starting point, the chapter delves into semantic description and explains the development of a Web API build using the REST paradigm to access cloud services pragmatically.

Learning Objectives

1. Understand the limitations of describing cloud services in natural language.
2. Understand how cloud services are programmatically accessed using a Web API.
3. Use Linked USDL and semantics technologies to describe cloud services.

(continued)

S. Stadtmüller (✉) • M. Junghans
Karlsruhe Service Research Institute (KSRI), Karlsruhe Institute of Technology (KIT),
Karlsruhe, Germany
e-mail: steffen.stadtmueller@kit.edu

J. Cardoso
Department of Informatics Engineering, Universidade de Coimbra, Coimbra, Portugal
Huawei European Research Center (ERC), Munich, Germany
e-mail: jcardoso@dei.uc.pt; jorge.cardoso@huawei.com

© Springer International Publishing Switzerland 2015
J. Cardoso et al. (eds.), *Fundamentals of Service Systems*, Service Science: Research
and Innovations in the Service Economy, DOI 10.1007/978-3-319-23195-2_5










Facebook	developers.facebook.com	Facebook data can be connected with countless business apps for endless possibilities.	
Twitter	dev.twitter.com	Enables applications to send and read tweets.	
YouTube	developers.google.com/youtube/	YouTube Data and Player API allow programmers to access functionality and content.	
Dropbox	dropbox.com/developers/	Grants access to data from external applications for easy revisions, file sharing, and search.	
Instagram	instagram.com/developer/	Enables applications to access user, relationship, media, comments, etc. of pictures and videos uploaded.	
LinkedIn	developer.linkedin.com	Companies can search for new employees and advertise job offerings.	
Wordpress	codex.wordpress.org/WordPress.org_API/	Enables companies to apply new functionality and features to WordPress.	
Lastfm	last.fm/api/	Clients can access information about albums, events, users, and artists.	
Google	developers.google.com/products/	Google API allowing access to their products' data such as maps, social networking, and email.	

Fig. 5.1 The most well-known companies contributing to the API economy

4. Use graph patterns to describe REST services.
5. Develop search algorithms which leverage semantic service descriptions.

► Opening Case Cloud Services foster the API economy

ACCESS TO CLOUD SERVICES VIA WEB API TECHNOLOGIES IS PROVIDING A NEW FORM OF INNOVATION

The API economy is an economy where companies providing cloud services expose their data assets to third parties using Web-accessible Application Programming Interfaces (Web API). Many well-known companies are already taking advantage of this new movement by exposing their businesses through a Web API, including: Facebook, Twitter, YouTube, Dropbox, and Instagram (Fig. 5.1).

The emergence of Web API technologies goes back to the year 2000. The first mover was salesforce.com who offered application interfaces to its clients. The largest impact was made by ebay.com who offered a Web API to the general public.

In the past, a technique called web scraping was the way developers programmatically retrieved data from web pages. It consisted of developing and customizing a web client to parse the HTML pages of a targeted website to extract formatted data.

```

1 from tweepy.streaming import StreamListener
2 from tweepy import OAuthHandler
3 from tweepy import Stream
4
5 consumer_key=""
6 consumer_secret=""
7 access_token=""
8 access_token_secret=""
9
10 class StdOutListener(StreamListener):
11
12     def on_data(self, data):
13         print data
14         return True
15
16     def on_error(self, status):
17         print status
18
19 if __name__ == '__main__':
20     l = StdOutListener()
21     auth = OAuthHandler(consumer_key, consumer_secret)
22     auth.set_access_token(access_token, access_token_secret)
23
24     stream = Stream(auth, l)
25     stream.filter(track=['music'])

```

Fig. 5.2 Access to the Twitter Web API to retrieve tweets with the term “music”

This process was complex, laborious, expensive, and required the reprogramming of client applications each time the provider made a change to the structure of web pages. With the emergence of Web API technologies, access to remote data became extremely simple as shown in Fig. 5.2 using, e.g., Python.

Two trends deserve to be mentioned: internal Web API and API Mashups.

Internal Web API While many interfaces are made publicly visible, a trend is for companies to also start using Web API technologies internally to capitalize on important collections of data assets.

Web API Mashups The use of Web API technologies enables third parties to mash data and functionality from various providers to create and deliver new products and services in response to emerging demand.

► Opening Case

5.1 Semantics in Cloud Services

As already explained in Chap. 2, cloud computing and cloud services are computing solutions based on the Internet. In the past, companies would run applications on computers physically located in their building, cloud computing allows users to

access the same types of applications over the Internet. Google Docs and Dropbox are well known examples of cloud services.

The landscape of cloud computing is expanding rapidly in size, diversity, and heterogeneity. Amazon AWS Marketplace,¹ which was launched in 2012, has up to now more than 1250 cloud services available. Noor et al. [1] found that almost 6000 cloud services are already available on the web. The study carried out consisted of searching the web using a customized cloud service crawler engine to find websites that offered cloud services. More than half a million links were parsed. Websites were individually analyzed using a text mining classifier because these cloud services were described in natural language. Additionally, ProgrammableWeb,² a popular registry for web-based services lists over 10,000 interfaces to different services. There is an exponential growth of the service ecosystem on the web that can be seen over the last years.

The increasing size of the cloud computing landscape brings several challenges. One of the difficulties is related to search since cloud services are described using web pages in natural language at providers' websites. As an example, Fig. 5.3 shows LastFM (last.fm) and its Web API (last.fm/api). LastFM is a SaaS instance providing a large online music catalog and recommendation platform. Its Web API provides programmers access to all data stored by the service. Web search engines like Google, Yahoo, or Bing work relatively well for finding popular articles on a specific subject using keywords. However, when search engines based on keyword matching are applied to the search of a specific type of cloud service, they fail to provide adequate results. In fact, all currently available search tools suffer either from poor precision (i.e., too many irrelevant documents) or from poor recall (i.e., too few relevant documents are found). For example, using the Google search engine to find a cloud redundant storage service with a cost per TB lower than \$25 per month, will return roughly 125,000 matching documents, but unfortunately it is not possible to immediately see any that matches the query. The results are littered with many other types of cloud services (e.g., "Amazon EC2 computing"), news articles on cloud services (e.g., "Top tips for doing a cloud storage cost analysis"), and reports (e.g., "How AWS pricing works").

This chapter describes how semantics can be used to address problems like the service search by enriching the description of cloud services using formal languages that enable computers to automatically interpret the description of the cloud services. Section 5.2 introduces semantic technologies that can be applied to create effective machine-readable service descriptions, as well as foundational design principles of Web APIs. Section 5.3 illustrates how Linked USDL [2], a semantic description language for services, can be used to enrich the description of a cloud service. Finally, Sect. 5.4 explains how formal descriptions can be explored by algorithms to support tasks such as search, matching, and ranking.

Throughout this chapter S-LastFM is used as an example for a cloud service with a Web API that leverages semantic technologies. S-LastFM is derived from LastFM, but offers slightly different functionalities and employs a formal language

¹ Amazon AWS Marketplace <http://www.aws.amazon.com/marketplace>.

² ProgrammableWeb <http://www.programmableweb.com>.

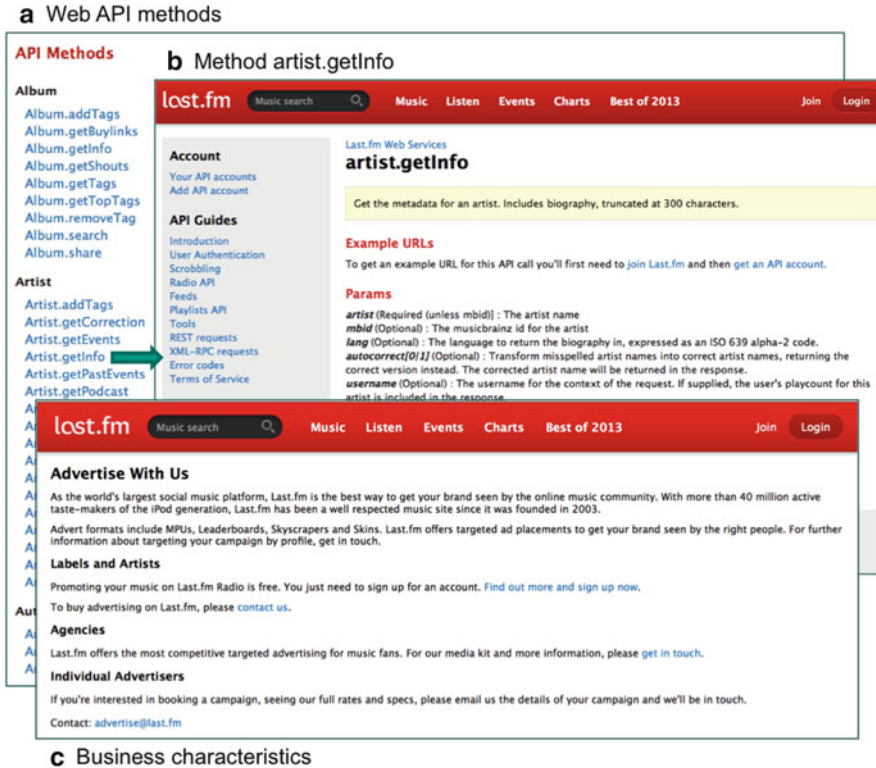


Fig. 5.3 The natural language description of technical (a and b) and business characteristics (c) of LastFM

to describe these functionalities: For the purpose of this chapter S-LastFM supports via its API the lookup of information of music artists, as well as the order of tickets for upcoming concerts. Further S-LastFM supports two interfaces that allow to search for such concerts.

5.1.1 Comprehensive Descriptions

Cloud services are complex entities that are not exhaustively characterized with descriptions for technical access via a Web API. In fact, pricing models, legal aspects, and service levels are elements that often need to be described explicitly when dealing with cloud services. While in the past the description of web services was mainly done at the technical level. Service description languages like WSDL [3] define the syntactical structure of service functionality descriptions. So-called semantic service description languages and frameworks like semantic annotations for WSDL (SA-WSDL) [4] and the semantic markup for web services OWL-S [5]

allow functionality descriptions that can be interpreted by machines. However, cloud services are more complex entities that also require the description of business and operational information.

- ▶ **Definition (WSDL)** The Web Service Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information (<http://www.w3.org/TR/wsdl>).

Business Perspective

A business-sensitive perspective represents a paradigm shift from the IT view on cloud services to a commercial view. For example, LastFM includes a natural language description of legal terms and policies, a pricing model, contact information, and a network of partners. This information is expressed in the form of natural language published as web pages, which makes it difficult for a computer to interpret it. Table 5.1 provides a set of examples extracted from the LastFM website. Section 5.3 explains how Linked USDL is used to include this information in structured descriptions.

Operational Perspective

The operational perspective is concerned with the elementary operations that a service provides. For example, LastFM provides more than 140 operations classified in 14 categories. Figure 5.3 shows the web page that displays all the operations, which are part of the LastFM Web API. These operations can be orchestrated to develop more advanced and complex cloud services, which can in turn be sold to consumers. Formally capturing this information using Linked USDL is also described in Sect. 5.3.

Technical Perspective

Cloud services are programmatically accessed using a Web API, an application-to-application communication technique. A Web API enables businesses to integrate

Table 5.1 The business perspective of LastFM

Charact.	Description	Web page (last.fm)
Terms of use	Terms that customers agree to follow to use the website. It also includes the definition of what it is considered to be “Acceptable Use”	/legal/terms
Privacy policy	A legal document that discloses some or all of the ways LastFM gathers, uses, discloses, and manages customers’ data	/legal/privacy
Subscription terms	The terms that subscribers agree to when they purchase and use their LastFM subscription	/legal/subscription

third party cloud services, such as a customer relationship management system (e.g., SugarCRM³), with a minimal effort into their own local software systems. The technical perspective indicates, which web standards can be used to interact with cloud services. For example, the Web API provided by LastFM is supported by REST services. Although the approach to describe a Web API in natural language (see Fig. 5.3a, b) is the one most often used, it is far from adequate for the goal of increasing the degree of automation of tools managing cloud services. Section 5.2 looks into the technological approaches to construct a Web API and Sect. 5.3.3 explores how they can be enriched and described using semantic knowledge in the form of graph patterns.

5.1.2 Cloud Service Tasks

Enriching cloud service descriptions usually does not only provide more comprehensive technical access to, and understanding of, the offered functionality, but also enables a higher degree of automation in performing high level tasks. Examples include search, composition, comparison, and clustering of cloud services [6]. Section 5.4 explains how algorithms that make use of semantic service descriptions can be developed to execute high level tasks, specifically, the task of service search, which consists of:

- Matching a service request with cloud service descriptions.
- Ranking available services according to their degree of match with a request.

Search is particularly interesting as it often serves as the foundation of tasks related to cloud services, e.g., the development of mashups, which are characterised as applications that combine content from several sources on the web. Mashups are often focused on an ad-hoc integration of involved cloud services. In order to build mashups, an identification of suitable cloud services is necessary, which demands service search capabilities that go beyond simple keyword search. Additionally, semantic cloud service descriptions mitigate the challenge of the actual combination of cloud services [7–9], e.g., by providing a clear understanding of involved input and output data. Semantic descriptions mitigate the problem of integrating data from various cloud services within the mashups [10] as data integration based on Semantic Web Technologies is more robust than the development of individual data schema mappings between pairs of cloud services.

³<http://www.sugarcrm.com>.

5.2 Foundational Technologies

This chapter looks into the use of semantic technologies as a solution to describe cloud services from a business and technical perspectives. Therefore, this section provides a brief introduction to the technologies that are used, namely, web semantics and Linked Data. It also introduces a Web API architecture, which is broadly used to implement cloud services. The presented material is a prerequisite to understand the description of cloud services using Linked USDL and graph patterns in the following sections.

5.2.1 The Semantic Web

The World Wide Web Consortium (W3C) started to work on the concept of a Semantic Web with the objective of developing solutions for data integration and interoperability. The goal was to develop ways to allow computers to interpret (sometimes termed understand) information on the web. The Semantic Web identifies a set of technologies and standards that form the basic building blocks of an infrastructure that supports the vision of a meaningful web.

- ▶ **Definition (Semantic Web)** The Semantic Web, as defined by the W3C (<http://www.w3.org/standards/semanticweb/>), is an extension of the classical *web of documents* and describes a set technologies and standards with regard to common data formats and exchange protocols to support a *web of data*.

Linked USDL is a fixed service description schema that was formalized using two technologies from the Semantic Web: the Resource Description Framework (RDF) [11] and RDF Schema (RDFS). RDFS was used to define a schema and vocabulary to describe services. This schema is applied to create RDF graphs that describe individual services. Both, RDF and RDFS, are used by applications that need to interpret and to reason over the meaning of information instead of just parsing data for display purposes.

- ▶ **Definition (RDF)** The Resource Description Framework (RDF) is a standard model for data interchange on the web (<http://www.w3.org/RDF/>).

RDF

The Resource Description Framework was developed by the W3C to provide a common data model enabling the description of information that can be read and interpreted by computer applications. RDF provides a graph model for describing resources on the web. A resource is an element (document, web page, printer, user, etc.) on the web that is uniquely identifiable by a universal resource identifier (URI).

A URI serves as a means for identifying abstract or physical resources. For example, <http://wikipedia.org/wiki/Metallica> identifies the location of a web page

about the band Metallica, and the following encoding `urn:isbn:1-420-09050-X` identifies a book using its ISBN. The RDF model is based upon the idea of making statements about resources in the form of a subject-predicate-object expression, a triple in RDF terminology. Each element has the following meaning:

Subject is the resource; the “thing” that is being described.

Predicate is an aspect about a resource that expresses the relationship between the subject and the object.

Object is the value that is assigned to the predicate.

RDF is based on a directed graph data model: A set of nodes are connected by (directed) edges. Nodes and edges are labeled with identifiers (i.e., URIs) that make them distinguishable from each other and allow for the reconstruction of the original graph from the set of triples. RDF offers a limited set of syntactic constructs—only triples are allowed.

Every RDF document is equivalent to an unordered set of triples, which describe a graph. For example, the RDF triple that describes the statement: “Metallica is the artist of Garage Inc.” is shown in Listing 5.1.

```
⌋ http://s-last.fm/album/garageInc , http://s-last.fm/artist ,
    http://s-last.fm/artist/metallica
```

Listing 5.1 Example of an RDF triple

The subject, `http://s-last.fm/album/garageInc`, is a resource representing a particular album. This resource has the property `http://s-last.fm/artist` with the value `http://s-last.fm/artist/metallica`. The statement can also be graphically represented as depicted in Fig. 5.4.

RDF blank nodes are used to express statements about individuals with certain properties without denominating the individual. The anonymity of blank nodes ensures that nothing besides the existence of the node can be inferred. Blank nodes, as the name suggests, may only occur in the subject or object position of a triple.

RDF literals describe data values that may only occur as property values. They are represented as strings and a shared interpretation is assumed to be given. Therefore, literals can be typed with a data type, e.g., using the existing types from the XML Schema specification [12]. Untyped literals are interpreted as strings.



Fig. 5.4 An example of an RDF graph

Turtle Syntax

While RDF is a data model, there are several serialization formats that can represent RDF graphs. Originally, XML was proposed and has been widely adopted by RDF data processing and management tools. Note that the data model is not affected by the choice of any of the serialization formats; the graph structures remain unchanged. Turtle, the Terse RDF Triple Language, is one of the serializations. It is a compact syntax for RDF that allows for the representation of graphs in natural text form [13]. It will be used in the remainder of this chapter.

In Turtle, every triple is completed by a full stop. A URI is represented in angle brackets and literals are enclosed in quotation marks. White spaces outside identifiers and literals are ignored. One way to represent the RDF statement from Fig. 5.4 using Turtle is shown in Listing 5.2.

Turtle permits abbreviations to further increase readability. For example, multiple triples with the same subject or triples with same subject and predicate can be pooled as shown in Listing 5.3 for an extended example.

The first lines introduce prefix abbreviations of the namespaces used. `rdf:type` is a property to state that the resource `sfmalbum:garageInc` is an instance of the class `sfm:Album`.⁴ The property `rdf:type` is often abbreviated to `a`. Capital first letters are used to indicate class names in contrast to individual and property names. The description of the location of the concert `sfmconcert:663239` makes use of a blank node representing the location resource. The location resource is not named but specified by its geographic coordinates embraced by square brackets.

```
1 <http://s-last.fm/album/garageInc> <http://s-last.fm/artist>
   <http://s-last.fm/artist/metallica> .
```

Listing 5.2 Turtle syntax representation of the RDF graph in Fig. 5.4

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix sfm: <http://s-last.fm/> .
3 @prefix sfmalbum: <http://s-last.fm/album/> .
4 @prefix sfmartist: <http://s-last.fm/artist/> .
5 @prefix sfmconcert: <http://s-last.fm/concert/> .
6 @prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
7
8 sfmalbum:garageInc sfm:artist sfmartist:metallica ;
9   rdf:type sfm:Album .
10 sfmartist:metallica rdf:type sfm:Artist .
11
12 sfmconcert:663239 a sfm:Concert ;
13   sfm:location [
14     geo:lat "48.7932" ;
15     geo:long "9.2258"
16   ] .
```

Listing 5.3 Turtle syntax representation of an RDF graph using abbreviations

⁴Compare to the collection resources of a REST architecture introduced in Sect. 5.2.2.

```
1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:sfm="http://s-last.fm/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
3   <sfm:Album rdf:about="http://s-last.fm/album/garageInc">
4     <sfm:artist>
5       <sfm:Artist
6         rdf:about="http://s-last.fm/artist/metallica"/>
7     </sfm:artist>
8   </sfm:Album>
</rdf:RDF>
```

Listing 5.4 RDF/XML serialization of the RDF graph from Listing 5.3

Many tools have been recently developed to support users in modeling structured data. Knowledge can be described with the support of ontology modeling tools like Protégé.⁵ A traditional text editor can also be used to create service descriptions, but dedicated applications, such as TextMate for Mac, provide syntax highlighting for Turtle, auto-completion, syntax validation, and format conversions. All helpful features that facilitate the modeling task. RDF graphs can be validated against a schema and converted to different serialization formats (including RDF/XML, Turtle, and others) with web-based tools like validators^{6,7} and translators [14]. Listing 5.4 shows the RDF graph of Listing 5.3 using the RDF/XML serialization.

RDF Schema

RDF Schema is a vocabulary language for RDF and allows the modeling of simple ontologies [15]. RDFS describes the logic dependencies among classes, properties, and values. While RDF provides universal means to encode facts about resources and their relationships, RDFS is used to express generic statements about sets of individuals (i.e., classes). RDFS associates the resources with classes (as shown in Listing 5.3), states the relations between classes, declares properties, and specifies the domain and range of properties.

- ▶ **Definition (RDFS)** RDF Schema (RDFS) provides a data-modelling vocabulary for RDF data. RDF Schema is an extension of the basic RDF vocabulary (<http://www.w3.org/TR/rdf-schema/>).

Classes in RDFS are much like classes in object oriented programming languages. They allow resources to be defined as instances of classes (by using the property `rdf:type`) and subclasses of classes. Subclass hierarchies can be specified by the RDFS property `rdfs:subClassOf`. The intuitive set theoretic semantics of class instances and subclasses (defined as member-of- and subset-of-relationships, respectively) ensures the reflexivity and transitivity of `rdfs:subClassOf`. The semantics of RDFS are specified in a W3C Recommendation [16].

⁵Protégé ontology editor and knowledge-base framework <http://www.protege.stanford.edu>.

⁶<http://www.rdfabout.com/demo/validator/>.

⁷<http://www.w3.org/RDF/Validator/>.

```
1 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3
4 sfm:artist rdf:type rdfs:Property ;
5   rdfs:subPropertyOf sfc:contributor ;
6   rdfs:label "Album artist"@en ;
7   rdfs:domain sfm:album ;
8   rdfs:range sfm:artist .
9
10 sfmalbum:garageInc sfm:artist sfmartist:metallica ;
11   sfm:released "1998-11-23"^^xsd:date .
```

Listing 5.5 Specification of domain and range of properties in RDFS

Properties can be seen as attributes that are used to describe the resources by assigning values to them. RDF is used to assert property-related statements about objects, and RDFS can extend this capability by defining the class domain and the class range of such properties.

As the example shown in Listing 5.5 indicates, property hierarchies can be specified with the RDFS property `rdfs:subPropertyOf`. A complete overview over the language concepts is provided in [12]. Literals, as shown in line 6 of Listing 5.5, describe data values for properties. A language tag, such as `@en` for English, is used to specify the language of the literal. Data type information can also be appended to literals following the double caret (cf. line 11). Each data type is also identified by its URI, which in turn allows applications to interpret their meaning.

The adoption of semantic technologies in the context of services implies an increased modeling and development effort. The development of appropriate domain ontologies, if required, can be time consuming. In order to mitigate this burden, different ontology learning approaches can automatically compute an ontological representation of the domain, e.g., from given text documents describing the domain [17, 18].

Given the logical statement nature of the knowledge represented with ontologies, traditional relational databases are not the ideal storage and query platform for RDFS. Knowledge is represented as sets of subject-predicate-object-triples and these are most efficiently stored and accessed in dedicated triple stores, such as Jena TDB⁸ and AllegroGraph.⁹ Likewise, querying triple stores is done via specific query languages: the current standard language for querying RDF(S) is SPARQL [19].

⁸Jena TDB <http://www.jena.apache.org/documentation/tdb/index.html>.

⁹AllegroGraph <http://www.franz.com/agraph/allegrograph/>.

SPARQL

The RDF information encoded is readable and interpretable by machines, e.g., software programs that utilize the knowledge in applications like a concert ticket selling application. SPARQL is an SQL-like query language that allows the retrieval of data from RDF graphs. Answers are computed by matching patterns specified in a query against the given RDF graph.

► **Definition (SPARQL)** The Simple Protocol and RDF Query Language Protocol (SPARQL) is a query language for RDF (<http://www.w3.org/TR/rdf-sparql-query/>).

Basic graph patterns are used in SPARQL queries when a set of triple patterns is matched. Listing 5.6 shows the SPARQL graph pattern query syntax. In SPARQL, Turtle is used to describe the graph patterns. In this example of a query, the set of artists, i.e., the individuals of the class `sfm:Artist`, are retrieved and returned.

The answer of `SELECT` queries are bindings for the variables (denoted with a question mark) listed directly after the keyword `SELECT`. In the example, the query results in variable bindings for `?artist`, which comprises, as shown in Table 5.2, a list of three artists represented by their URI as used in the RDF graph. Other query forms, e.g., `ASK`, `DESCRIBE`, and `CONSTRUCT`, allow queries for other kinds of information. `ASK` returns a boolean answer about the existence of a solution for a specified graph pattern. A `DESCRIBE` query returns an RDF graph describing specified resources.

SPARQL is also able to build RDF graphs and return them as results. The example from Listing 5.7 shows how the `CONSTRUCT` query form can be used to list places in which bands have played. Therefore, this query specifies a template used to build a graph-based on the matching results of the query.

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX sfm: <http://s-last.fm/>
3
4 SELECT ?artist
5 WHERE
6 {
7   ?artist rdf:type sfm:Artist .
8 }
```

Listing 5.6 SPARQL query to retrieve instances of the class `Artist`

Table 5.2 Results of the SPARQL query shown in Listing 5.6

Artist
<code><http://s-last.fm/artist/metallica></code>
<code><http://s-last.fm/artist/rihanna></code>
<code><http://s-last.fm/artist/eminem></code>

```
1 CONSTRUCT { ?artist sfm:playedIn ?place }
2 WHERE
3 {
4   ?artist rdf:type sfm:Artist .
5   ?artist sfm:event ?event .
6   ?event geo:located ?place .
7 }
```

Listing 5.7 SPARQL query asking for artists and the places they have performed at

Linked Data

Linked Data is a subset of the Semantic Web that adheres to the principles of the Semantic Web architecture: commitment to the use of RDF(S) and URIs to denote things. In particular, Linked Data adheres to the following four design principles [20]:

- Use URIs to name “things”.
- Use HTTP URIs so that people can lookup the names.
- Lookups on those URIs provide further information describing the “things” in RDF.
- Include links to other URIs in the descriptions to allow people to discover further “things”.

The use of an HTTP URI allows machines and humans to lookup the name and get useful information about resources adhering to the RDF and SPARQL standards. The HyperText Transfer Protocol (HTTP) is prevalently used to exchange data on the web.¹⁰ The use of an HTTP URI further guarantees the uniqueness of the identifier.

The resolvable resource description should contain links to other resource identifiers so that users can discover more things.¹¹ Linkage comprises external and internal links (for any predicate) and the reuse of external vocabularies, which can be interlinked. The special property `owl:sameAs` specifies the equivalence of different identifiers that refer to the same thing. For example, the band Metallica is described in different vocabularies or websites. Overlapping data of different sources can be aligned by equivalence statements as illustrated in Listing 5.8.

Adhering to the Linked Data principles has many advantages that apply in the context of structured representation of data on the web but also in the context of the formal description of services. For example, for service search, selection, composition, and analysis.

¹⁰See IETF RFC7230 at <http://tools.ietf.org/html/rfc7230> et seq. for details.

¹¹Linked Data—Design Issues <http://www.w3.org/DesignIssues/LinkedData>.

```

1 @prefix owl: <http://www.w3.org/2002/07/owl#> .
2
3 <http://dbpedia.org/resource/Metallica> owl:sameAs
  <http://s-last.fm/artist/metallica> .

```

Listing 5.8 Establishing the equivalence of resources using the property owl : sameAs

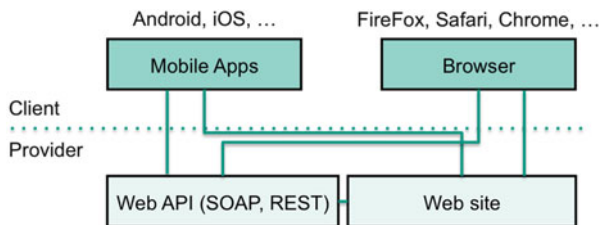


Fig. 5.5 Architecture of services provided over a web API and described using web pages

5.2.2 Web API Design Principles

Application Programming Interfaces (API) are used to provide general access to the functionality of a services. Client applications like mobile applications (apps) or web pages containing interactive web forms can make use of an API to offer these functionalities to end users. The resulting layered architecture depicted in Fig. 5.5 provides the advantage that applications using the Web API can be developed by the service provider as well as third party developers, thus increasing the potential reach of the service.

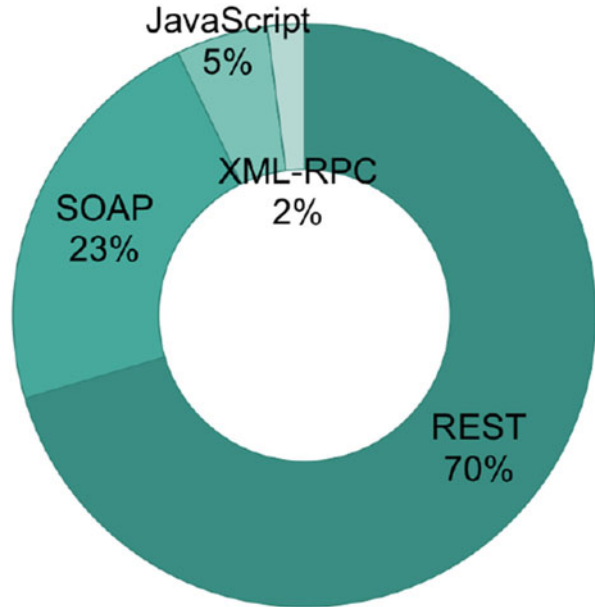
For example, the provider of S-LastFM offers the service functionality with an API described at s-last.fm/api. The S-LastFM homepage makes use of this API to offer the services to the consumers. Additionally, third party developers can develop mobile or web applications to provide the services of S-LastFM on mobile devices. According to a W3C note,¹² when constructing a Web API two general alternatives for the implementation exist:

- An API that exposes an arbitrary set of operations.
- A resource-oriented API with a uniform set of stateless operations.

Both types correspond to the types of cloud services described in Chap. 2. This section only gives a brief overview of operations oriented approach and focuses on the architecture of resource-oriented interfaces, since the latter are predominantly used on the web (see Fig. 5.6): A detailed comparison of both API paradigms is available from [21, 22].

¹²<http://www.w3.org/TR/ws-arch/#relwwwrest>.

Fig. 5.6 Protocol usage of public web interfaces according to ProgrammableWeb



ProgrammableWeb.com 12.09.2013

Operation-Oriented API

An API with arbitrary operations is often designed by adhering to the variety of specifications and languages commonly referred to as the WS-* stack. However, other variants exist, e.g., remote procedure calls. The main characteristic of this style of API is that the operations that make out the service are directly defined and offered. When implemented for cloud services such an API uses the web as a transport layer for the data and entails a high degree of freedom. For example, the S-LastFM API can offer the following operations. Client applications can invoke these operations to invoke the desired functionalities.

- `findArtistInfo` using the name of an artist as input.
- `findConcerts` using an identifier of concerts as input.
- `orderConcertTicket` using an identifier of the concert as input.

Resource-Oriented API

A resource-oriented API complies to the constraints of a Representational State Transfer (REST) architecture [23]. According to the Richardson maturity model [24], REST is identified as the interaction between a client (i.e., an application) and a server based on three principles:

1. The use of URI-identified resources.
2. The use of a constrained set of operations, i.e., the HTTP methods, to access and manipulate resource states.

3. The application of hypermedia controls, i.e., the data representing a resource contains links to other resources. Links allow a client to navigate from one resource to another during interaction.

According to statistics from ProgrammableWeb, the majority of publicly accessible interfaces are designed according to REST principles (cf. Fig. 5.6). Therefore, this chapter will elaborate in the following on the concepts of REST in more detail.

Resources

A REST API offers the service functionality under the primacy of resources rather than operations. A resource can be a real world object or a data object on the web. Resources are uniquely identified with a URI. The representation of a resource details its current state, i.e., relevant information associated with the resource. Resources can be grouped into collections. These collections in turn are also URI-identified resources themselves (so called *collection resources*). The resulting tree-like structure of the resources is similar to the well known directory structure of file systems. In the case of the S-LastFM API, collection resources representing artists, concerts, etc. correspond to the folders of a file system, while the concrete instances of artists and concerts correspond to files put into the corresponding folders. In order to illustrate this concept, Table 5.3 lists examples of instances and collection resources of the S-LastFM Web API.

Constrained Operations Set

The interaction of client applications with cloud services via a REST API is not based on the call of API-specific operations but rather on the direct manipulation of exposed resource representations or the creation of new resource representations. A manipulation of the state representation implies that the represented resource is manipulated accordingly. For such manipulations, REST offers only a constrained set of operations that can be applied to a resource. These operations are shared by all interfaces following REST principles, which increases interoperability and understandability of the interfaces. Nevertheless, not all resources must necessarily allow the application of all possible methods.

Some of the methods can carry input data as a payload, which describes the intended new state of the addressed resource.¹³ On the web, the allowed operations are the HTTP methods (cf. Table 5.4) and the identifier of a resource is an HTTP URI, which makes the web the underlying platform for the API. For example, S-LastFM uses the following methods:

- GET. Almost all resources of S-LastFM allow for the application of GET to retrieve information. A GET on one of the collection resources gives an overview of the known instances of concerts and artists. The retrieval of the information

¹³The HTTP POST method is a noteworthy exception as it permits the submission of data to process, which is similar to an RPC call and therefore should be used carefully.

Table 5.3 Resources of the S-LastFM web API

URI/description
http://s-last.fm/artist A collection resource of music artists with a representation that contains links to all artists
http://s-last.fm/artist/metallica http://s-last.fm/artist/rihanna http://s-last.fm/artist/eminem Resources representing individual music artists. The representation contains information about the artist (e.g., name and genre) and links to upcoming concerts
http://s-last.fm/concert A collection resource of concerts with links to all concerts
http://s-last.fm/concert/1234 http://s-last.fm/concert/1235 http://s-last.fm/concert/1236 Resources representing individual concerts with information about the concert (e.g., location, date, and price per ticket) and links to the performing artist and to the <code>ticketorder</code> resource
http://s-last.fm/ticketorder Collection resource of ticket orderings with a representation containing links to all ticket orderings
http://s-last.fm/ticketorder/567 http://s-last.fm/ticketorder/568 http://s-last.fm/ticketorder/569 Resources representing individual ticket orderings with information about the ordering (e.g., delivery and address) and link to corresponding concert

Table 5.4 Overview of HTTP methods (excerpt)

Method	Safe	Idempotent	Description
GET	✓	✓	Retrieve the current state of a resource
OPTIONS	✓	✓	Retrieve a description of possible interactions
DELETE		✓	Delete a resource
PUT		✓	Create or overwrite a resource with the submitted input
POST			Send input as subordinate to a resource or submit input to a data-handling process

of a specific ticket order however is only allowed with the correct credentials. Only a user who created the ordering can look it up again. The retrieval of the collection resource for all orderings is generally not permitted.

- POST. The collection resource for ticket orderings allows POST to enable users to add a new ticket order for a specific concert.
- PUT. S-LastFM allows PUT only to overwrite existing ticket orders, to enable users to update an order.
- DELETE. Ticket orders can also be canceled with DELETE up to a predefined time before the concert takes place.

Two types of methods exist: safe and unsafe. Safe methods guarantee not to affect the current states of resources, while unsafe methods change the state of the resources. Furthermore, most of the methods are idempotent. The repeated application of an idempotent method on a resource does not change the state of the resource beyond the first application of the method. For example, if a client application deletes a resource, deleting it again has no effect.

Hypermedia Controls

A REST API fosters loose coupling between clients and services on the premise that client applications do not need to know about all resources in advance. The retrievable representations of some known resources contain links to other resources that the client can discover during runtime. Applications can use such discovered resources to perform further interaction steps. Collection resources specifically contain links to all the resources in the collection. This architectural design allows client applications to be robust toward changes in the API, because the application has to react to whatever it finds when it interacts with the API [24, 25].

For example, at the behest of an end user a client application can interact with the resources of S-LastFM in the following way:

1. Retrieve the information (i.e., representation) of a specific artist, which contains links to upcoming concerts (method GET).
2. Retrieve the information of one of the concerts, which contains a link to the collection of ticket orderings (method GET).
3. Create in this collection a new resource representing a new ticket order (method PUT).

Note that only the identifier of the resource representing the artist needs to be known in advance (i.e., before the interaction can start). This could be further refined by providing a search interface over the available artists as contained in the artist collection resource.

In a REST architecture, no constraints are given on how the status of a resource has to be represented. There is no defined standard regarding data model or serialization format of the data that detail the current state of a resource or the input and output data of a method. Client application and API are, however, supposed to agree on the format of the exchanged data and implicitly on how the data is supposed to be interpreted. The process of establishing this agreement is called *content negotiation*. For different application scenarios such an agreement requires vendor specific content types (i.e., content types defined by the service provider) for the individual services to convey the meaning of the communicated data. The idea behind vendor specific content types is that service providers can reuse content types and application developers can make use of specific content type processors in their applications to work with the data. In practice, however, most Web API providers simply make use of standard non-specific content types, e.g., `text/xml` or `application/json` [26]. Developers therefore have to write applications that are individually adapted for the API they make use of. Section 5.3.3 details another solution to this problem based on Linked Data resource representations.

5.3 Linked USDL

The need for descriptions at the business and operational levels redirected efforts to the development of new languages to capture these perspectives beside the technical one. USDL [27], the Unified Service Description Language, is probably the most comprehensive attempt. It provides a shared semantic vocabulary¹⁴ to describe services.

- ▶ **Definition (USDL and Linked USDL)** The Universal Service Description Language (USDL) is a platform-neutral language for describing services. Linked USDL is a remodeled version of USDL build upon the Linked Data principles.

The initial USDL development started in 2008 to describe business, software, and real world services using a computer-understandable specification to make them available on the web. The Internet of Services (see Chap. 2) requires services to be traded and, thus, places emphasis on the description of business characteristics such as pricing, legal aspects, and service level agreements. This was the main motivation to create USDL. In 2012, a new version named Linked USDL [28] was developed based on Linked Data principles and represented with RDFS. This is the version which will be explored in this chapter.

5.3.1 Linked USDL Family

Linked USDL is segmented into modules. The objective of this division is to reduce the overall complexity of service descriptions by enabling providers to only use the modules needed. Currently, five modules exist:

`usdl-core` The core module covers concepts central to a service description. It includes operational aspects such as interaction points between the provider and consumer that occur during provisioning and the description of the business entities involved.

`usdl-price` The pricing module provides a range of concepts which are needed to adequately describe price structures in the service industry.

`usdl-agreement` The service level agreement module gathers functional and non-functional information on the quality of the service provided, e.g., availability, reliability, and response time.

`usdl-sec` This module aims to describe the main security properties of a service. Service providers can use this specification to describe the security goals, profiles, and mechanisms.

¹⁴A vocabulary is also known called a “schema”, a “data dictionary”, or an “ontology”.

`usdl-ipr` This module captures the usage rights of a service. Rights are often associated with the concept of copyright which implicitly exists with the creation of an artifact (i.e., a service) and does not need to be registered with an office in order to be granted.

For example, customers and providers can use `usdl-agreement` [29] to create service level agreements to monitor whether the actual service delivery complies with the agreed service level. In case of violations, penalties or compensations can be directly derived. The module `usdl-ipr` is fundamental for business environments to define rights and obligations when consuming a service. The official Linked USDL website is available at linked-usdl.org and the current Linked USDL modules, use cases, and documentation are available at GitHub – <http://www.github.com/linked-usdl/>.

5.3.2 The Core Module

The Linked USDL Core can be regarded as the center of the Linked USDL family since it ties together all aspects of service description distributed across the USDL modules. Figure 5.7 shows the conceptual diagram on the core module. Classes are represented with an oval, while properties with an edge. Linked USDL Core has 12 classes (e.g., `Service`, `ServiceOffering`, `InteractionPoint`, and `Role`) and 13 properties (e.g., `receives`, `hasInteractionPoint`, `withBusinessRole`, and `hasInteractionType`).

To explain how a service can be described with Linked USDL, S-LastFM will be used as an example. Most of the information used for the modeling was retrieved from the LastFM web site and shown in Fig. 5.3. The description was written using the Turtle language.

Before starting with the formal representation of the S-LastFM service, a set of prefixes needs to be defined to ease the modeling with Turtle. Listing 5.9 shows the prefixes that refer to the Linked USDL modules which will be used in this running example (lines 1–5). Other prefixes required, but not shown, refer to vocabularies such as Dublin Core¹⁵ (shown in the example with `:dcterms`) and GoodRelations¹⁶ (`:gr`). The extract also shows the standard heading statements required to create a service instance (lines 7–13).

```

1 @prefix usdl: <http://linked-usdl.org/ns/usdl-core#> .
2 @prefix usdl-br:
   <http://linked-usdl.org/ns/usdl-business-roles#> .
3 @prefix usdl-it:
   <http://linked-usdl.org/ns/usdl-interaction-types#> .

```

¹⁵The Dublin Core Schema is a vocabulary that can be used to describe web resources (video, images, web pages, etc.), as well as physical resources such as books.

¹⁶GoodRelations is a vocabulary for product, price, store, and company data that can be embedded into web pages to be automatically processed by intelligent applications.

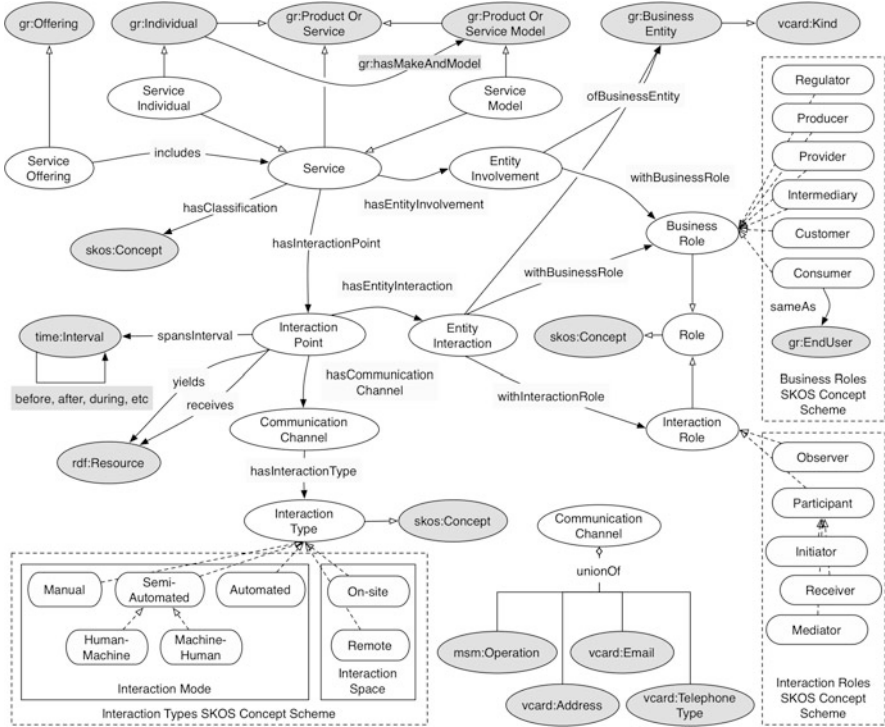


Fig. 5.7 The linked USDL core schema [28]

```

4 @prefix usdl-ir:
  <http://linked-usdl.org/ns/usdl-interaction-roles#> .
5 @prefix usdl-pr: <http://linked-usdl.org/ns/usdl-pricing#> .
6 ...
7 <> a owl:Ontology ;
8   rdfs:label "S-LastFM service description" ;
9   dcterms:title "S-LastFM service description" ;
10  dcterms:description "Enhanced description of last.fm, a
  popular music recommendation service." ;
11  dcterms:created "2014-01-17"^^xsd:date ;
12  dcterms:modified "2014-01-18"^^xsd:date ;
13  owl:versionInfo "001" .

```

Listing 5.9 Prefixes and standard statements for the S-LastFM service description

The Service

The next step is to specify the actual service to be described. The class `usdl:Service` provides the entry point for the description. As shown in Listing 5.10, the new service was named `service_SLastFM`. The specification also includes:

```

1 :service_SLastFM a usdl:Service ;
2   gr:description "A semantic recommendation service for
   music." ;
3
4   usdl:hasServiceModel :model_SLastFM ;
5   usdl:hasEntityInvolvement [
6     a usdl:EntityInvolvement ;
7     usdl:ofBusinessEntity :be_SLastFM_Ltd ;
8     usdl:withBusinessRole usdl-br:provider
9   ] ;
10  usdl:hasInteractionPoint :ip_Advertise ;
11  usdl:hasInteractionPoint :ip_TicketOrder .

```

Listing 5.10 The S-LastFM service class

- Associating a service model with the service.
- Specifying the business entities participating during service provisioning.
- Enumerating the interaction points provided by the service.

The class `usdl:ServiceModel` is used to create groupings of services that share a number of characteristics. For example, a service model for S-LastFM can group services characterized by supplying online music. In the same line of thought, the service “Vodafone unlimited internet service” may belong to the grouping “Internet provisioning service”. The example from Listing 5.10 associates the service `service_SLastFM` with the service model `model_SLastFM`¹⁷ using the property `usdl:hasServiceModel` (line 4).

The class `usdl:EntityInvolvement` captures the `gr:BusinessEntity` involved in the service delivery and the `usdl:Role` they play (lines 5–9). Entity involvement is a fundamental construct introduced in Linked USDL to enable capturing service networks [30]. This enables specifying, for instance, that a given music service is provided by a certain company or that a third party is involved in the service delivery chain. The properties `usdl:ofBusinessEntity` and `usdl:withBusinessRole` enable to specify the business entities involved during the provisioning of a service and also the role that each one takes. In Listing 5.10, the business entity is defined with the class `be_SLastFM_Ltd` and its role is defined as `usdl-br:provider`. Linked USDL provides a default taxonomy of basic business roles that cover the most typical ones encountered during service modeling such as regulator, intermediary, producer, and consumer. The prefix `usdl-br` identifies the taxonomy `usdl-business-roles`¹⁸ which defines the default roles available.

¹⁷The definition of the service model `model_SLastFM` is not provided in this running example.

¹⁸<http://www.linked-usdl.org/ns/usdl-business-roles>.

```

1 :be_SLastFM_Ltd a gr:BusinessEntity ;
2   foaf:homepage <http://slast.fm/> ;
3   foaf:logo
4     <http://cdn.last.fm/flatness/badges/lastfm_red.gif> ;
5   gr:hasISICv4 "5920"^^xsd:string ;
6   gr:hasNAICS "512220"^^xsd:string ;
7   gr:legalName "SLast.fm Ltd."^^xsd:string ;
8   gr:taxID "830 2738 46"^^xsd:string ;
9
10  vcard:hasAddress
11    [ a vcard:Work ;
12      vcard:country-name "UK"@en ] .

```

Listing 5.11 Description of the business entity providing the S-LastFM service

Listing 5.11 illustrates the description of the company providing the S-LastFM service with the class `be_SLastFM_Ltd`. The description includes the ISIC (International Standard Industrial Classification of All Economic Activities) code for S-LastFM: 5920 – sound recording and music publishing activities. It also specifies the NAICS (North American Industry Classification System) code, legal name, tax ID number, and country where the company is located (lines 5–8).



The North American Industrial Classification System (NAICS) and the International Standard of Industrial Classification (ISIC) codes are six digit numbers used to describe the primary economic activity of businesses. They classification codes can be used for looking up reports on specific industries or to find companies within a certain industry.

Interaction Points

The extract from Listing 5.10 also defines two interaction points, `ip_Advertise` and `ip_TicketOrder`, for the service `service_SLastFM`. An interaction point (`usdl:InteractionPoint`) represents an actual step in performing the operations made available by a service. On a personal level, an interaction point can model that consumer and provider need to meet in person to exchange service parameters or resources during service delivery (e.g., documents that are processed by the provider). On a technical level, this can translate into calling a web service operation (most cloud services are accessed and invoked via web services). An interaction point can be initiated by the consumer or the provider. Since interaction points may be long running and have an ordering with respect to other interaction points, they are subclasses of `TimeSpanningEntity`. It is, therefore, possible to express temporal relationships between interaction points such as “before” or “after”. For richer expressions, the time ontology constructs can be used.

Listing 5.12 shows the interaction point `ip_Advertise` which enables customers to book advertising campaigns and inquire about rates and specs. Interaction points define four main pieces of information:

```

1 :ip_Advertise a usdl:InteractionPoint ;
2   dcterms:title "S-LastFM Advertisement"@en ;
3   dcterms:description "If you are interested in booking a
      campaign, seeing our full rates and specs, please send
      us the details of your campaign and we will get in
      touch."@en ;
4
5   usdl:hasCommunicationChannel [
6     a usdl:CommunicationChannel ;
7     vcard:country-name "UK";
8     vcard:locality "London";
9     vcard:postal-code "SE1 0NZ";
10    vcard:street-address "Last.fm Ltd., 5-11 Lavington
      Street" ;
11    usdl:hasInteractionType usdl-it:manual ;
12    usdl:hasInteractionType usdl-it:remote
13  ];
14
15  usdl:hasCommunicationChannel [
16    a usdl:CommunicationChannel ;
17    vcard:hasEmail <mailto:advertise@slast.fm> ;
18    usdl:hasInteractionType usdl-it:manual ;
19    usdl:hasInteractionType usdl-it:remote
20  ];
21
22  usdl:hasEntityInteraction [
23    a usdl:EntityInteraction ;
24    usdl:withBusinessRole usdl-br:provider ;
25    usdl:withInteractionRole usdl-ir:participant
26  ];
27
28  usdl:hasEntityInteraction [
29    a usdl:EntityInteraction ;
30    usdl:withBusinessRole usdl-br:customer ;
31    usdl:withInteractionRole usdl-ir:participant
32  ];
33
34  usdl:receives dbpedia:Advertising ;
35  usdl:yields dbpedia:Contract .

```

Listing 5.12 An interaction point involving human interaction

- The communication channels that customers or applications can use to interact with a service.
- The entities that are involved during the realization of the interaction point.
- The resources that are needed for an interaction point to execute properly.
- The resources that are generated from the execution of an interaction point.

The example identifies two types of communication channels (lines 5–20):

- `vcard:Address`
- `vcard:Email`

These classes are not shown in the example since they are all grouped by the class `usdl:CommunicationChannel`.

Communication Channels

Communication channels are additionally characterized by their interaction type. Linked USDL provides two default taxonomies covering the main modes (i.e., automated, semi-automated, or manual) and the interaction space (i.e., on-site or remote). The example describes how customers can ask for information to advertise a campaign with S-LastFM. This can be done by using “snail” mail or email. All the communication channels of the example require a manual (`usdl-it:manual`) and remote (`usdl-it:remote`) interaction. This means that humans, not software applications, will be involved in the interaction with S-LastFM.

The example also indicates the role of the two entities that will interact (lines 22–32): both the provider and customer are participants. This information is represented using the class `usdl:EntityInteraction` which links interaction points to business entity types (e.g., provider, intermediary, and customer) using the property `usdl:withBusinessRole` and the role they play within the interaction (e.g., initiator, mediator, and receiver) using the property `usdl:withInteractionRole`.

Receives and Yields

Listing 5.12 shows that the interaction point receives (`usdl:receives`) and yields (`usdl:yields`) resources (lines 41–42). Receives is the input required and yields corresponds to the outcome produced by an interaction point. The example shows that the interaction point `ip_Advertise` receives a resource identified by the concept `dbpedia:Advertising` and yields a resource identified by the concept `dbpedia:Contract`. Since `dbpedia` is the computer-understandable formal mirror of `wikipedia`, the use of these two properties reach twofold objectives:

- Provide a natural language description via wikipedia.com which is suitable to be interpreted by humans.
- Provide a formal description via dbpedia.org which is suitable to be processed by software applications.

Naturally, other computer-processable data sources such as freebase.com can also be used.

Automated Communication Channels

While the previous example of an interaction point involved only human participants, the example from Listing 5.13 illustrates a fully automated interaction which

```
1 :ip_TicketOrder a usdl:InteractionPoint ;
2   dcterms:title "Ticket Orders"@en ;
3   dcterms:description "Collection of all ticket orders;
4     allows to submit new ticket orders"@en ;
5
6   usdl:hasCommunicationChannel [
7     a sfm:TicketOrder ;
8     usdl:hasInteractionType usdl-it:automated ;
9     usdl:hasInteractionType usdl-it:remote ;
10  ] ;
11
12  usdl:hasEntityInteraction [
13    a usdl:EntityInteraction ;
14    usdl:withBusinessRole usdl-br:provider ;
15    usdl:withInteractionRole usdl-ir:participant ;
16  ] ;
17
18  usdl:hasEntityInteraction [
19    a usdl:EntityInteraction ;
20    usdl:withBusinessRole usdl-br:consumer ;
21    usdl:withInteractionRole usdl-ir:initiator ;
22    usdl:withInteractionRole usdl-ir:receiver ;
23  ] ;
24
25  usdl:receives sfm:Concert ;
26  usdl:yields sfm:TicketOrder .
```

Listing 5.13 An interaction point for an application-driven interaction

does not require human intervention (lines 5–9). This means that on both sides of the communication channel, applications will be involved during service provisioning by exchanging data. This requires a well-defined API which must be understood by applications.

The most well-known approaches to describe services semantically are OWL-S, WSMO, SAWSDL, and WSMO-Lite when it comes to SOAP Web services, and MicroWSMO and SA-REST for REST Web services [28]. In order to cater for interoperability, Linked USDL Core uses what can essentially be considered the maximum common denominator between these formalisms: the minimal service model.¹⁹ The MSM is a simple RDFS vocabulary to capture the semantics of SOAP and REST Web services in a common model.

The resources exposed in a REST architecture can be seen as communication channels of the service. However, the hypermedia constrains (cf. Sect. 5.2.2) of REST imply that clients are supposed to discover many of the resources by following links. Therefore, not all the exposed resources need to be described and covered by a Linked USDL description: only the resources a client is expected to start its interactions with need to be included as interaction points in a Linked USDL

¹⁹<http://www.iserve.kmi.open.ac.uk/ns/msm/>.

description. Additional resources will be discovered. In the case of S-LastFM, the resource `sfm:TicketOrder` can be considered an entry point for an interaction. In contrast, the individual resources representing the artists or concerts are not necessary to include in the Linked USDL description, since clients do not start their interactions with these resources directly.²⁰

Listing 5.14 shows how the operation `sfm:TicketOrder` can be semantically described. In this example, the class `msm:Operation` is extended with the property `gpd:graphPattern` to provide a link to graph pattern-based descriptions (explained in Sect. 5.3.3) that detail the exchanged data when ordering a concert ticket by interacting via the communication channel. This new property is attached to the only given part of the message content of input and output respectively.

Service Offering

A `usdl:ServiceOffering` is an offer made by a `gr:BusinessEntity` of one or more instances of `usdl:Service` to customers. An offering usually associates a price, legal terms of use, and service level agreement with a service. In other words, it makes a service a tradable entity. Listing 5.15 illustrates an offering named `offering_SLastFM` for the service `service_SLastFM` (lines 1 and 11). Such a service offering may have limited validity over geographical regions or time, the offering adds various pieces of information such as temporal validity, eligible regions, and accepted payment methods (lines 2–10).

Finally, the last part of the example indicates that the classes `legal_SLastFM` and `price_SLastFM` describe the legal aspects and the price of the S-LastFM service, respectively (lines 12–13).²¹

```

1 sfm:TicketOrder a msm:Operation ;
2   msm:hasInput _:input ;
3   msm:hasOutput _:output ;
4   hrest:hasMethod "POST"^^xsd:string .
5
6 _:input a msm:MessageContent ;
7   msm:hasPart _:in1 .
8 _:in1 gpd:graphPattern <http://s-last.fm/ticketorder/input> .
9
10 _:output a msm:MessageContent ;
11   msm:hasPart _:out1 .
12 _:out gpd:graphPattern <http://s-last.fm/ticketorder/output> .

```

Listing 5.14 Example of a communication channel of type `msm:Operation` from Listing 5.13

²⁰Since clients might require descriptions for all resources to interact with them, not included resources have to provide similar information directly, which can be retrieved at runtime for example via the HTTP OPTIONS method.

²¹The description of the legal and price modules are not covered in this chapter.

```

1 :offering_SLastFM a usdl:ServiceOffering ;
2   gr:validFrom "2014-01-24T00:00:00+01:00"^^xsd:dateTime ;
3   gr:validThrough "2015-12-24T00:00:00+01:00"^^xsd:dateTime ;
4   gr:eligibleRegions "DE"^^xsd:string, "US-CA"^^xsd:string ;
5   gr:acceptedPaymentMethods gr:VISA,
6     gr:ByBankTransferInAdvance ;
7   gr:eligibleDuration [
8     a gr:QuantitativeValue ;
9     gr:hasValueInteger "1"^^xsd:int ;
10    gr:hasUnitOfMeasurement "MON"^^xsd:string
11  ] ;
12  usdl:includes :service_SLastFM ;
13  usdl:legal :legal_SLastFM ;
14  usdl:price :price_SLastFM .

```

Listing 5.15 A concrete offering of a service

5.3.3 Technical API Descriptions

A Linked USDL description of an application web service with a remote and automated communication channel also entails a description on the technical level, besides the business and operation levels. Such technical information can be used by clients for the invocation of the service or the automation of high level tasks (see Sect. 5.4). This section explains how technical information can be provided for REST architectures, where resources are represented with Linked Data.

Service Resource States

For resource-driven or REST-based services, the technical description of the API does not have to be focused on the operations. Since REST architectures only use a constrained and shared set of operations, the intuition or meaning of the operations is always inherently clear and well defined. For example, in the case of a REST API based on HTTP, the semantics of the available operations²² (cf., Table 5.4) are defined in an RFC specification²³ and therefore do not need to be explicitly described further in the service description.

A REST architecture offers the functionality of a service by allowing the states of resources to be changed. It is these possibilities to manipulate a resource that need to be described. The manipulation of a resource state is the result of an application of an unsafe method at a resource, where the payload (i.e., input data) details the desired state change. The output data that is returned after the application of a method details the new state of the affected resource [31]. Therefore, to describe the manipulation possibilities of a resource, it is sufficient to detail the possible inputs

²²The operations of an HTTP-based REST API are also referred to with the term *HTTP methods*.

²³See IETF RFC7230 at <http://tools.ietf.org/html/rfc7230> et seq. for details

and the resulting outputs for the available operations of the resource. Furthermore, the relation between the input and output (i.e., how the input influences the resulting state) needs to be made explicit.

In a REST architecture, client application and server are supposed to form a contract with content negotiation, not only on the data format but implicitly also on the semantics of the communicated data, i.e., an agreement on how the data has to be interpreted [25]. Since the agreement on the semantics is only implicit, programmers developing client applications have to manually gain a deep understanding of the provided data and the manipulation possibilities, often based on natural text descriptions. However, the reliance on natural language descriptions for the interfaces lead to mashup designs in which programmers are forced to write glue code with little or no automation and to manually consolidate and integrate the exchanged data.

Linked Data (see Sect. 5.2.1 “Linked Data”) unifies a standardized interaction model with the possibility to align vocabularies using RDF, RDFS, and OWL. However, the interactions are currently constrained to simple data retrieval. Following the motivation to look beyond the exposure of fixed data sets, the extension of Linked Data with REST technologies has been explored and many approaches recognize the value of combining REST services and Linked Data [32–35]. These approaches propose to use Linked Data to represent the state of a resource, and as a format for input and output data.²⁴ The use of unique identifiers as central elements of Linked Data mitigates the problem of the only implicitly negotiated semantics of the communicated data (see Sect. 5.2.2). Furthermore, Linked Data is already focused on the interlinkage of resources, which makes the design of a hypermedia control-driven API architecture straight forward.

Graph Pattern Descriptions

Since Linked Data uses a graph-based data model, many existing approaches [33–35] propose graph patterns as a means to describe the input and output data of possible resource manipulations. The use of graph patterns enable to automatically recognize the vocabularies that are understood by an API and detail how the data representing the resource states is structured. In the scenario of ordering a concert ticket via S-LastFM a user has to (mediated by an application) create a new order resource. An example of the state of an order resource (e.g., identified by <http://s-last.fm/ticketorder/567>) expressed with Linked Data is shown in Listing 5.16.

The state representation of the resource details the price of the tickets, the amount, and the estimated time until the tickets are delivered (lines 11–13). Additionally, the resource contains links to the user who ordered the tickets and to the concert (lines 8 and 10). Following the link to the resource representing the user can, e.g., bring results regarding the delivery address or credit card details for payment. Some of the elements in the state representation are different for every individual order resource. These variable elements are either provided by the user that submits the order via the client application or they are set by S-LastFM directly:

²⁴If another format like JSON or XML is used, a description needs to make the implicit semantics of the data explicit.

```

1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix sfm: <http://s-last.fm/> .
3 @prefix sfmorder: <http://s-last.fm/ticketorder> .
4 @prefix sfmuser: <http://s-last.fm/user> .
5 @prefix sfmconcert: <http://s-last.fm/concert> .
6
7 sfmorder:567 rdf:type sfm:Ticketorder .
8 sfmorder:567 sfm:user sfm:JohnDoe .
9 sfm:JohnDoe rdf:type sfm:User .
10 sfmorder:567 sfm:event sfmconcert:1234 .
11 sfmorder:567 sfm:price "120 USD"@en .
12 sfmorder:567 sfm:amount "2" .
13 sfmorder:567 sfm:eta "10 days"@en .
14 sfmconcert:1234 rdf:type sfm:Concert .

```

Listing 5.16 An example of a state representation of an order resource

```

1 Method: POST
2 Input:
3
4 _:order rdf:type sfm:Ticketorder .
5 _:order sfm:user ?u .
6 ?u rdf:type sfm:User .
7 _:order sfm:amount ?a .
8 _:order sfm:event ?c .
9 ?c rdf:type sfm:Concert .

```

Listing 5.17 Input data in the form of a graph pattern

- **Provided by the user.** For example, the user name `sfm:JohnDoe` and the amount of the ordered tickets `"2"`.
- **Provided by S-LastFM.** For example, the URI of the newly created order resource `sfmorder:567` and the price of the tickets `"120 USD"`.

Input Graph Pattern

To create a new ticket order, a client application has to POST data (see Listing 5.17) detailing the desired new resource to the collection resource of the ticket orders. A graph pattern-based description for the input data of a POST to the ticket order collection resource (`http://s-last.fm/ticketorder`) uses variables for the elements that are different for every individual order resource. The client has to use a blank node `_:order` for the input data, since it is not possible to know in advance what identifier the API will assign to the new resource.

Output Graph Pattern

The returned data after the application of POST characterizes the state of the created resource (see Listing 5.18). Therefore, the description of the output data is similar to the input description, but contains, additionally, variables for the information added by the API (e.g., the price of the ordered tickets).

```
1 Method: POST
2 Output:
3
4 ?order rdf:type sfm:Ticketorder .
5 ?order sfm:user ?u .
6 ?u rdf:type sfm:User .
7 ?order sfm:amount ?a .
8 ?order sfm:event ?c .
9 ?c rdf:type sfm:Concert .
10 ?order sfm:price ?p .
11 ?order sfm:eta ?t .
```

Listing 5.18 Output data in the form of a graph pattern

Input and output descriptions share common variables (e.g., `?u`, the user that ordered the tickets). The use of the same variable implies that it will be bound by the same value. This establishes a connection between input and output descriptions and allows to describe the relation between both.

Finally, it is important that input as well as output descriptions of resources can be treated as URI-identified resources as well:

- `http://s-last.fm/ticketorder/input` as URI for the input description
- `http://s-last.fm/ticketorder/output` as URI for the output description

Since the descriptions are URI-identified resources themselves, it is possible provide links to the descriptions. Such links can be used as a response when the `HTTP OPTIONS` method is applied to the collection resource of the ticket orders, which allows a client to look up the descriptions. The links are also used to refer to the graph pattern descriptions in the USDL document detailing a communication channel of the service (cf. Sect. 5.3.2).

5.4 Service Search Algorithm

Descriptions for services can be utilized to increase the degree of automation of higher level tasks, such as service search, composition and discovery. This section details how graph pattern-based descriptions, which can be provided as part of a Linked USDL description, can be leveraged for *service search* [36]. Given a specific functionality that is required to be fulfilled by a service (e.g., the order of tickets for concerts), the process of searching for a service refers to the problems of

- Matching, i.e., identifying the subset of services that are suitable to provide the functionality out of a potentially large set of services
- Ranking, i.e., comparing the identified services in terms of their capability to satisfy the given requirements.

However, service descriptions can be leveraged to improve the automation of other tasks such as the invocation of a service or the composition of several services to create a complex service.

5.4.1 Matching

In a scenario where services are described with Linked USDL, which provides links to graph pattern-based input and output descriptions, service search requests can be formulated as *service templates*. These templates follow the same syntax as the graph pattern descriptions. Therefore a service template consists of two graph patterns that represent a template of:

- all possible *input* RDF graphs a client can provide for the invocation of a service,
- the *output* RDF graphs such an agent expects to be delivered by the service.

Similar to the graph pattern descriptions a service template details a clients possibilities to provide input for a service operation. Further the template describes the requirements of the client on the output of a service operation, as well as the expected relation between the input and output.

Therefore, the question of whether a given service description matches a service template correlates to the problem of graph pattern containment. The input graph pattern of a service description must be contained in the service template's pattern. This containment relation implies that every graph that satisfies the template input graph pattern must also satisfy the service description's input graph pattern. Intuitively, the input a client is able to provide fulfils the requirements of the service to be invoked. Note that this also allows for a client to provide additional data, which it can provide for service invocation even though a matching service does not require them.

A client can for example express in an input template that it can provide information about a location and a music genre. If a service details in an input description that it requires just the information about the location, template and description are matching, since the description is contained in the template. However, if a description details that information about location and an artist is required, template and description are not matching.

Matching the output graph patterns works in an analogous way. The output graph pattern of a service description contains the output graph pattern of a template, which implies that every graph that satisfies the service description output graph pattern also satisfies the template output graph pattern. So the required containment relation of the output patterns is dual to that of the input graph patterns. Intuitively again this means a service output has to provide enough to satisfy the request, but can provide more.

A client can for example express in an output template that it is looking for information from a service about a concert. If a service details in an output description

that it provides information about concerts and genre, template and description are matching, since the template is contained in the description. However, if template of the client details it requires information about concerts and ticket prices, template and description are not matching.

Formulas 5.1 and 5.2 summarize the relations between the patterns in a service description and a service template in a search process.

$$\textit{Service Description Input Pattern} \subseteq \textit{Template Input Pattern} \quad (5.1)$$

$$\textit{Template Output Pattern} \subseteq \textit{Service Description Output Pattern} \quad (5.2)$$

5.4.2 Ranking

The matching based on graph pattern containment consists of two binary decisions (one for the input and one for the output), of whether a service description completely matches a service template. However, it is sensible to assume that often services only partly satisfy the requirements of an agent, or that an agent has not all the necessary data for the invocation. Therefore, services should be ordered according to the degree they match to a given request. To allow for such a flexible search approach a ranking of service descriptions against service templates has to be enabled with continuously-valued matching metrics, e.g.:

- The *predicate subset ratio* (*psr*) measures to what degree the set of predicates used in one pattern are subsumed within the set used in another.
- The *resource subset ratio* (*rsr*) measures to what degree the set of named resources, in subject or object position, used in one pattern are subsumed with those of another pattern.
- The *containment ratio* (*cr*) measures to what degree triple patterns in one graph pattern are contained in the other pattern.

The metrics *psr* and *rsr* indicate to what degree a service description and a service template are using the same vocabulary. These vocabulary-based metrics allow to test whether a service description and a template use some of the same resources and predicates (and to what degree). Therefore, they provide a mechanism to discover services, which are close to a given template, but are not necessarily completely matching.

Similarly to the pattern containment, metrics for input and output have to be distinguished. A template input graph pattern can offer more data than actually needed by a service without endangering their compatibility. Therefore, the subset ratios for the input patterns have to measure to what degree the resources (respectively predicates) in the service descriptions are used in the service template. For the subset ratios of the output patterns this works the other way around, because a service can

offer more output than required by the template. In this case the subset ratios have to measure to what degree the resources (respectively predicates) in the template are used in the service description. The Eqs. (5.3) to (5.6) show how the metrics are calculated.

$$psr_{input} = \frac{|P_{templ} \cap P_{decsr}|}{|P_{decsr}|} \quad (5.3)$$

$$psr_{output} = \frac{|P_{templ} \cap P_{decsr}|}{|P_{templ}|} \quad (5.4)$$

$$rsr_{input} = \frac{|R_{templ} \cap R_{decsr}|}{|R_{decsr}|} \quad (5.5)$$

$$rsr_{output} = \frac{|R_{templ} \cap R_{decsr}|}{|R_{templ}|} \quad (5.6)$$

where,

- P_{templ} the set of all predicates in the template.
- P_{decsr} the set of all predicates in the service description.
- R_{templ} the set of all resources in the template.
- R_{decsr} the set of all resources in the service description.

Let the input description pattern use the predicates `rdf:type`, `sfm:performer` and `sfm:location`. The input template uses `rdf:type`, `sfm:performer` and `sfm:artist`. Then $p_{input} = 2/3$, because only two out of three predicates in the description are used in the template.

The metric cr measures to what degree a graph pattern P_{sub} is contained in another pattern P_{super} . To realize this, the measurement cr is based on the power set $\mathcal{P}(P_{sub})$ of triple patterns derived from the graph pattern P_{sub} . The largest set of triple patterns $T_{max} \in \mathcal{P}(P_{sub})$ (i.e., the set with the most triple patterns) in the power set is identified, that is contained in P_{super} :

$$T_{max} \subseteq P_{super} \quad (5.7)$$

Note that T_{max} is not necessarily unique in $\mathcal{P}(P_{sub})$. However, just one of the largest triple pattern sets that are contained in P_{super} needs to be identified for the calculation of cr . Equation (5.8) shows how the metric cr is calculated. The metric describes the ratio between the number of triple patterns in the identified set T_{max} and the overall number of triple patterns in the original pattern P_{sub} , from which the power set is derived.

$$cr = \frac{|T_{max}|}{|P_{sub}|} \quad (5.8)$$

cr measures precisely to what degree a graph pattern is subsumed by another, thus expressing the containment degree of one pattern in relation to another.²⁵ Note, that if a graph pattern is completely contained in another one (i.e., $cr = 1.0$), the subset ratios must necessarily result in a metric of 1.0.

The vocabulary-based metrics do not regard the structure of the graph patterns to match service descriptions and templates. The metrics rsr and psr are therefore less precise search metrics compared to cr , i.e., even if all vocabulary-based subset ratios result in a value of 1.0, it is not guaranteed that a service description and a template match in terms of pattern containment. On the other hand, if $cr = 1.0$ for the input and output patterns, a complete match is guaranteed. However, the vocabulary-based metrics are computationally less expensive to calculate. To achieve a scalable service search system rsr and psr can be used to filter service descriptions for a given template: if either psr or rsr result in a value of 0.0, it can be inferred without additional calculation that the value of cr has to be 0.0. Furthermore, other low values for rsr and psr can be used as thresholds. If one of the vocabulary-based metrics falls below the defined threshold, the calculation of cr can be omitted. The setting of this threshold depends on the required reaction time of the service search system, as well as the desired values for precision and recall.

Figure 5.8 and Table 5.5 illustrate an example of a search process in the S-LastFM scenario. In the example a client is searching for a functionality where the client can interact with a resource to acquire information about upcoming concerts. In the input of the interaction with the resource the client can include information

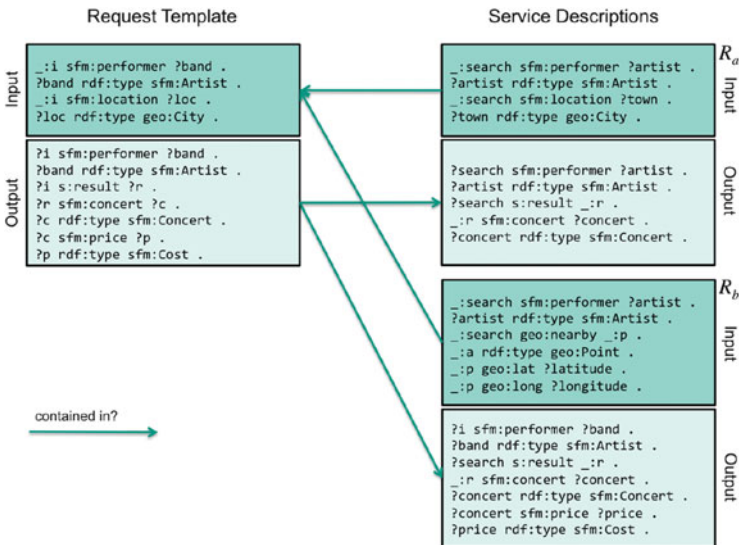


Fig. 5.8 An example of the ranking process

²⁵For a detailed description of how to calculate the described metrics see [36].

Table 5.5 An example of the ranking process (cont.)

	Service description R_a		Service description R_b	
	Input	Output	Input	Output
Pattern containment	Yes	No	No	Yes
cr	1.0	0.71	0.33	1.0
psr	1.0	0.8	0.4	1.0
rsr	1.0	0.66	0.5	1.0

about the artist and the city in which the client is looking for concerts. As output of the interaction the client expects to find a concert and information about the price for tickets. The example assumes two descriptions are available for:

- Resource R_a to which requests can be submitted with artist and city. R_a provides output information about concerts (but no ticket price information).
- Resource R_b to which requests can be submitted including artist and a geographical point with latitude and longitude. R_b provides output information about concerts including ticket price information.

In this example the client has enough information to invoke the interaction with R_a , but this will not provide all the desired information, since no information about the price of concert tickets are delivered by R_a . Interacting with R_b would provide all the desired information, but the client does not have enough data to invoke it (the latitude/longitude coordinates of the city are missing). This is also reflected in the containment metrics as detailed in Table 5.5. It can be seen how the psr and rsr metrics can serve as an estimation for the more precise cr metric.

Therefore in the given example no perfect match for the template can be found. The client has to resolve this problem and decide:

- Resolution 1: Choose R_a and forgo some of the desired results.
- Resolution 2: Choose R_b and try to acquire missing information (e.g., with the help of another API).

In terms of ranking R_a would be preferred over R_b with regard to the input, but R_b would be preferred over R_a with regard to the output. If a definite order of the resources is required, a weighted average can be calculated from the input and output metrics. The weights reflect which of the resolution strategies is preferred by the client.

5.5 Conclusions

This chapter looked into how rich and comprehensive service descriptions can be constructed to facilitate the search of cloud services. Naturally, rich descriptions improve, not only search, but also other computational tasks such as service selection, discovery, classification, and composition.

The approach followed started by taking a cloud service Web API and used semantic web technologies to enrich and formally describe the Web API structure from a technological perspective as well as from a business perspective. The cloud service selected was LastFM since it is a popular service on the web. The formal description was done by relying on Linked USDL to add semantic knowledge. Linked USDL provides a business and technical envelope to describe services' general information and their Web API. To exemplify the benefits of using semantic service descriptions, an algorithm for service search was developed to demonstrate how semantics can improve precision and recall.

Review Section

Review Questions

1. What are the differences between interfaces and services? What is a Web API?
2. Why is a keyword-based search insufficient for the search for a services or an API?
3. What is the purpose of the operational perspective on services. Compare it to the technical perspective, which also includes a formal description of the operations.
4. Draw the RDF graph of the RDF document shown in Listing 5.13 on page 163.
5. Create a SPARQL query to retrieve the business and corresponding interaction roles of all the entities involved in an interaction from the RDF document given in Listing 5.13. The correctness of the query and the obtained results can be evaluated with existing tools such as Jena TDB or AllegroGraph.
6. Explain the concept of a collection resource?
7. Why does REST restrict the set of available interaction methods? Is this constraint too restrictive?
8. Which aspects of a service can be modeled in the business perspective of Linked USDL?
9. Name use cases in which structured information about the service provider can be useful.
10. What is the relationship between an interaction point and an operation in Linked USDL?
11. What is the main benefit of the use of hypermedia controls?
12. How is the state of a resource described? What kind of information is included?
13. Graph-patterns have been used to describe input and output data of an API. Is it possible to use RDF(S) descriptions instead?
14. In which cases can other relationships (than the ones introduced in Formulas 5.1–5.2 on page 170) between the patterns in service and service template descriptions be useful?

Project

The implementation of services is an extensive task. Besides traditional design, implementation, and testing of the software that already provides the service functionality, a number of further tasks have to be considered in order to effectively provide a functionality as a service. As this chapter conveys, the interfaces provide access to resources and methods of the underlying software systems. Further, a formal, i.e., machine-readable and machine-interpretable, description of the provided service is a prerequisite for managing large services automatically.

The use of formal service descriptions increases the degree of automation that is necessary when a large set of services are managed. Different tasks of management may require that information about different perspectives on services. Due to the importance of expressive and formal service descriptions, the goal of this project is to practice, apply, and gain a more detailed understanding on service descriptions. Within this project, we will continue to use Linked USDL as a service description model in combination with the Semantic Web technologies.

1. Find a service that is publicly available in the Internet. The service should expose at least a few number of different resources and methods, which are provided by a public Web API. This API is described in semi-structured web pages that describe resources and methods in natural language.
2. Make yourself familiar with the involved resources and methods. Identify two non-trivial methods of the API that will be formally described in the subsequent steps.
3. At this stage, a vocabulary of the domain of intercourse is needed. Create an RDFS vocabulary describing the resources of chosen methods. The vocabulary can be sketched in form of an RDF graph drawn on paper. Alternatively, existing modeling tools like Protégé or a simple text editor can be used alike.
4. Furthermore, the Linked USDL service description model is used to create a description for this service and its chosen methods. A complete overview on Linked USDL can be obtained from <http://www.linked-usdl.org>.
5. Create a Linked USDL description of the technical perspective of the service. This description should cover the methods and all the involved resources, like input and output parameters. Since Linked USDL service descriptions can be serialized to regular RDF documents, a plain text editor or any RDF modeling tool can be used. A dedicated Linked USDL editor is currently under development.
6. Extend this service description, such that the business and operative perspectives are added. The provider of the service describes legal terms, service quality, and other information related to these perspectives within web pages, which may contain large text blocks. The relevant information needs to be identified and expressed formally. The RDFS domain vocabulary can be extended accordingly, if required.

Within this project, it should become obvious how tedious it can be to deal with and to interpret natural language text descriptions of services. It is easy to imagine how difficult it can be to analyze, search, or invoke services from a large service repository. The need and the benefits of formal descriptions as well as the broad spectrum of perspectives gained by the use of Linked USDL were conveyed.

Key Terms

Web API The acronym API stands for application programming interface and enables applications to exchange data. A Web API is the web version of this interface. It is leveraging web technologies (e.g., URI, HTTP, HTML, JSON, and XML) to enable companies to make their data assets available to external developers.

Service Description A service description defines and characterizes the services offered to customers. It includes functional and non-functional characteristics (properties). Typically, services are described in natural language, but more modern approaches use semantic web technologies.

USDL The Unified Service Description Language (USDL) was the first comprehensive language to describe services using computer-understandable formats. It described aspects such as legal constraints, pricing models, service level, and interactions between customers and providers [27, 37].

Linked USDL Linked USDL is a simpler version of USDL which was designed using Linked Data principles. It is more adequate to work on web environments since it uses solely web protocols and specifications, and it describes services semantically [28].

Semantic Web The Semantic Web is a technology stack originally developed by the W3C which enables people to share knowledge beyond the boundaries of websites. The specifications which are part of the stack, and include URI, RDF(S), OWL, and SPARQL, strive to turn unstructured data into computer-readable formats.

Linked Data Linked Data is a paradigm which applies the principles and technologies of the web to share and link data. It can be viewed as a pragmatic way of applying the technology stack provided by the Semantic Web to build worldwide networks of linked knowledge.

Turtle Turtle (Terse RDF Triple Language) is a serialization for capturing knowledge expressed using the Resource Description Framework (RDF). It is relatively simple to read/write when compared to other serializations, e.g., XML.

Further Reading

Jorge Cardoso and Amit Sheth. *Semantic Web Services, Processes and Applications*. Springer, 2006.

Rudi Studer, Stephan Grimm, and Andreas Abecker. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2010.

Leonard Richardson, Mike Amundsen, and Sam Ruby. *RESTful Web APIs Paperback*. O'Reilly Media, 2013.

References

1. Noor T et al (2013) CSCE: a crawler engine for cloud services discovery on the world wide web. In: IEEE 20th international conference on web services (ICWS). IEEE, Washington, DC, USA, pp 443–450
2. Cardoso J, Pedrinaci C (2015) Evolution and overview of linked USDL. In: 6th International conference exploring services science (IESS 2015). Lecture notes in computer science. Springer, Berlin
3. Christensen E et al (2013) Web services description language (WSDL) 1.1. W3C note. W3C, Mar 2001. <http://www.w3.org/TR/wsdl>. Accessed 10 Sep 2015
4. Kopecky J et al (2007) SAWSDL: semantic annotations for WSDL and XML schema. IEEE Internet Comput 11(6):60–67
5. Martin D et al (2004) OWL-S: semantic markup for web services. W3C member submission
6. Cardoso J, Sheth A (2003) Semantic e-workflow composition. J Intell Inf Syst 21(3):191–225
7. Endres-Niggemeyer B (eds) (2013) Semantic mashups. Intelligent reuse of web resources. Springer, Berlin/Heidelberg
8. Le Phuoc D et al (2009) Rapid prototyping of semantic mash-ups through semantic web pipes. In: Quemada J et al (eds) Proceedings of the 18th international conference on world wide web, WWW 2009, Madrid, 20–24 April 2009. ACM, New York, NY, USA pp 581–590
9. Lathem J, Gomadam K, Sheth AP (2007) SA-REST and (s)mashups: adding semantics to RESTful services. In: Proceedings of the first IEEE international conference on semantic computing (ICSC 2007), Irvine, CA, 17–19 Sept 2007. IEEE, Washington, DC, USA, pp 469–476
10. Di Lorenzo G et al (2009) Data integration in mashups. SIGMOD Rec 38(1):59–66
11. Manola F, Miller E (2013) RDF primer. W3C recommendation. W3C, Feb 2004. <http://www.w3.org/TR/rdf-primer/>. Accessed 10 Sep 2015
12. Brickley D, Guha RV (2013) RDF vocabulary description language 1.0: RDF schema. W3C recommendation. W3C, Feb 2004. <http://www.w3.org/TR/rdf-schema/>. Accessed 15 Aug 2013
13. Beckett D, Berners-Lee T (2013) Turtle - terse RDF triple language. W3C team submission. W3C, Mar 2011. <http://www.w3.org/TeamSubmission/turtle/>. Accessed 10 Sep 2015
14. Stolz A, Rodriguez-Castro B, Hepp M (2013) RDF translator: a RESTful multi-format data converter for the semantic web. Technical report TR-2013-1, Universität der Bundeswehr München, July 2013
15. Staab S, Studer R (eds) (2009) Handbook on ontologies, 2nd edn. International handbooks on information systems. Springer, Berlin.
16. Hayes PJ (2013) RDF semantics. W3C recommendation. W3C, Feb 2004. <http://www.w3.org/TR/rdf-mt/>. Accessed 10 Sep 2015
17. Maedche A, Staab S (2001) Ontology learning for the semantic web. IEEE Intell Syst 16(2): 72–79
18. Lehmann J, Voelker J (2014) An introduction to ontology learning. In: Lehmann J, Voelker J (eds) Perspectives on ontology learning. AKA/IOS Press, Heidelberg, pp 9–16
19. W3C SPARQL Working Group (2013) SPARQL 1.1 overview. W3C recommendation. W3C, Mar 2013. <http://www.w3.org/TR/sparql11-overview/>. Accessed 10 Sep 2015
20. Bizer C, Heath T, Berners-Lee T (2009) Linked data - the story so far. Int J Semantic Web Inf Syst 4(2):1–22

21. Pautasso C, Wilde E (2009) Why is the web loosely coupled?: a multi-faceted metric for service design. In: Quemada J et al (eds) International conference on world wide web. ACM, New York, NY, USA
22. Stadtmüller S et al (2013) Comparing major web service paradigms. In: Workshop on services and applications over linked APIs and data, vol 1056. CEUR-WS, 2013
23. Fielding R (2000) Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine
24. Richardson L, Ruby S (2007) RESTful web services. O'Reilly Media. Sebastopol, CA, USA
25. Webber J (2010) REST in practice: hypermedia and systems architecture. O'Reilly Media, Farnham, UK
26. Maleshkova M, Pedrinaci C, Domingue J (2010) Investigating web APIs on the world wide web. In: IEEE 8th European Conference on Web Services (ECOWS), IEEE, Ayia Napa
27. Cardoso J et al (2010) Towards a unified service description language for the internet of services: requirements and first developments. In: IEEE international conference on services computing (SCC), FL, pp 602–609
28. Pedrinaci C, Cardoso J, Leidig T (2014) Linked USDL: a vocabulary for web-scale service trading. Lecture notes in computer science, vol 8465. Springer, Berlin, pp 68–82
29. García JM et al (2015) Linked USDL agreement: effectively sharing semantic service level agreements on the web. In: The IEEE international conference on web services (ICWS), New York, 2015
30. Cardoso J et al (2012) Open semantic service networks. In: International symposium on services science (ISSS), Leipzig, 2012
31. Stadtmüller S et al (2013) Data-Fu: a language and an interpreter for interaction with read/write linked data. In: International conference on world wide web. International world wide web conferences steering committee, Geneva, Switzerland, pp 1225–1236
32. Wilde E (2009) REST and RDF granularity. Accessed 10 Sep 2015
33. Krummenacher R, Norton B, Marte A Towards linked open services and processes. In: Future internet symposium. Springer, Berlin, pp 68–77
34. Speiser S, Harth A (2011) Integrating linked data and services with linked data services. In: Proceedings of the 8th extended semantic web conference (ESWC'11) part I, Lecture notes in computer science, vol 6643. Springer, Heraklion
35. Verborgh R et al (2011) Efficient runtime service discovery and consumption with hyperlinked RESTdesc. In: International conference on next generation web services practices, Salamanca, 2011
36. Stadtmüller S, Norton B (2013) Scalable discovery of linked APIs. *J Metadata Semant Ontol* 8(2):95–105
37. Cardoso J, Winkler M, Voigt K (2009) A service description language for the internet of services. In: International symposium on services science (ISSS), Leipzig, 2009