View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Servicio de Coordinación de Bibliotecas de la Universidad Politécnica de Madrid

A VOCABULARY FOR THE MODELLING OF IMAGE SEARCH MICROSERVICES

José Ignacio Fernández-Villamor, Carlos A. Iglesias, Mercedes Garijo
Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid
{jifv, cif, mga}@dit.upm.es

Keywords: Service description, mashups, semantic web, web applications, REST

Abstract: In order to take advantage of the services that are available on the Web, several approaches that allow describing services have been proposed. With them, developers can publish service descriptions, allowing services to be automatically executed and composed. However, in most cases, the service description task is not carried out, partly because it is a time-consuming task. This has caused initiatives such as WSMO lite, SA-REST, hRESTS or Microservices, that try to reduce complexity in services, to appear.
Also, an increasing number of web applications have followed the Linked Data initiative and publish information that is machine processable thanks to Semantic Web technologies such as RDF. However, sometimes direct access to information requires the usage of search forms and, in other cases, spidering techniques such as focused crawling in order to aggregate and filter data. Automatic execution of search services would improve access to information in the web by enabling agents to automatically aggregate, filter and directly access data.
In this paper, it is presented how the Microservices framework can provide a feature-based vocabulary for the description of image search services. Microservices framework is a lightweight service description framework that take feature-oriented and aspect-oriented programming ideas to service description. The article illustrates how this vocabulary can characterise a set of popular search services, such as Google Images or Flickr. In addition, the article describes how this vocabulary can be used for the development of new services, such as a metasearcher that aggregates results from various search services.

1 INTRODUCTION

In the current open Internet, several services are available in web applications, following the Service-Oriented Architecture (Erl, 2005) through either Web Services architecture (McIlraith et al., 2001) or REST architectural style (Fielding, 2000). This view suggests services being publicly accessible to make the Web a global software library, with software services and components ready for their use by developers.

In order to make services machine processable, Semantic Web Services (Zhou et al., 2006) standards such as OWL-S (World Wide Web Consortium, 2004) or WSMO (Roman et al., 2005) allow building semantic service descriptions. Machine agents can automatically process service descriptions to perform execution, discovery and composition operations in an automated way.

Web Services architecture misses integration with REST architectural style (Wilde and Gaedke, 2008), resulting in the use of the Web as a platform and thus adding unnecessary complexity to the protocol stack. In order to improve the integration of the service architecture with the REST architectural style of the Web, initiatives such as WADL (Hadley, 2006) try to enable Semantic Web Services approach with a RESTful design in mind.

Anyway, both WS-based and REST-based semantic web services approaches share the common target of defining semantically rich service descriptions in order to enable agents to perform automatic tasks such as service discovery, execution, and composition. Unfortunately, Semantic Web Services have not reached wide adoption yet (Murphy et al., 2008),

partly because building service descriptions is a very time-consuming task. This has originated research in lightweight semantic service descriptions with initiatives like WSMO lite (Vitvar et al., 2007), or RESTful approaches such as SA-REST (Sheth et al., 2007), hRESTS (Wright State University, 2008), or Microservices (Fernández-Villamor et al., 2010).

Also, semantic technologies such as RDFa (Adida and Birbeck, 2008) allow annotating web resources and publishing Linked Data (Berners-Lee, 2006), which makes information machine processable. Still, there are pieces of information which are either hidden behind search services (or, in general, any HTML form) or hardly accessible, so long as information discovery in the Web is performed through web spidering. This makes that tasks such as getting a particular archive of posts in a web log is a tough task that requires spidering techniques such as focused crawling (Chakrabarti et al., 1999). Therefore, by describing search services semantically, access to information would be improved.

This paper attempts to reduce the effort of building service descriptions of image search services through the use of lightweight semantics. We define a vocabulary for the description of image search services by using Microservices framework, a lightweight approach to service description inspired in aspect-oriented and feature-oriented programming paradigms. Microservices enable automatic discovery of service descriptions, as well as reusing feature descriptions among different services. A vocabulary for defining descriptions for image search services has been defined and applied on a case study of building a metasearcher that aggregates search results from different search services.

The article is structured as follows. First, the fundamentals of the Microservices framework is presented in section 2. Then, section 3 presents a vocabulary for describing image search services, obtained from the analysis of popular image search services. Section 4 describes how this vocabulary has been applied on a case study for building a metasearcher that aggregates search results from different search services. Then, section 5 describes related work. Finally, section 6 draws out the main conclusions and future work derived from this research.

2 MICROSERVICES FRAMEWORK

We will use Microservices framework (Fernández-Villamor et al., 2010), a lightweight RESTful feature-oriented service description framework for describing services in RESTful web applications. Microservices framework attempts to simplify the process of defining service descriptions to favour automatic service consumption in the Semantic Web. In this section, an introduction of the motivation behind this framework is given, followed by a description of it.

2.1 Feature-orientation

Microservices framework apply ideas inspired in mixins, aspect-oriented and feature-oriented programming paradigms to semantic service description. These paradigms extend object-oriented programming by allowing the modelling of secondary concerns in an isolated way. Feature-Oriented Programming (FOP) (Prehofer, 1997) is a composition model that allows refining classes through the definition of features, i.e., subclasses with core functionality. Mixins (Bracha and Cook, 1990), or abstract subclasses, are separate groups of methods that can be inserted into a class to override the original behaviour, but which cannot be instantiated on their own. Therefore, mixins serve to implement features in FOP (Apel et al., 2006). FOP can be seen as a generalization of traditional class inheritance in Object-Oriented Programming, and is used to develop the so-called software product lines, i.e. programs that provide different combinations of features (Lopez-Herrejon, 2005).

Aspect-Oriented Programming (AOP) (Elrad et al., 2001) similarly proposes separating concerns that cross-cut various classes or methods, with the logging aspect as the most popular example of a cross-cutting concern. Code from aspects is injected into specified join points in classes. This way, by using AOP, logging commands can be inserted into appropriate join points in a class without changing the original code of the class.

AOP and FOP are different paradigms despite the existing similarity between them, as pointed out in (Lopez-Herrejon, 2005), so long as they propose different methods to combine code (code weaving vs. modification of inheritance chains). Some efforts try to combine the two approaches by introducing new concepts such as Multi Mixins, Aspectual Mixins, and Aspectual Mixin Layers (Apel et al., 2005).

Feature and aspect orientation have inspired other modelling approaches, such as Feature-Oriented Model Driven Development (Trujillo et al., 2007), in which models are created by composing features. Similarly, Role-oriented programming (Steimann, 2000) or Subject-oriented programming (Harrison and Ossher, 1993) regard separation of object roles and the so-called subjective perceptions, respectively.

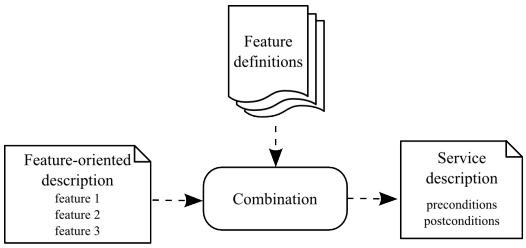

Figure 1: Feature-oriented description framework

In Microservices framework, the idea of separating features and concerns is employed to enable feature-oriented service modelling. A service description is a composition of features, allowing the reuse of feature descriptions in a similar way as features, aspects and abstract classes are reused in the previously mentioned paradigms. This modelling approach is described in the following section.

2.2 Modelling approach

Microservices framework follows a feature-oriented description framework, as shown in figure 1. This framework allows to describe services semantically in terms of their features in order to allow reusing descriptions at a feature level.

In Microservices framework, a service is modelled as a set of features f_1, f_2, ..., f_i that it has. In web applications, some examples of service features are "*performing a retrieval operation*", "*requiring user authentication*", "*performing a storage operation*", "*handling images*", or "*outputting a set of resources*".

In order to provide a mapping between a service's feature set and a service's formal description, Microservices framework includes feature definitions. A feature definition $\mathcal{F}$ is a rule that maps a set of features f_1, ..., f_k to a set of conditions. It is activated when the set of features is present in the service's feature-oriented description, and produces a formal description. By aggregating all descriptions from the activated feature definitions, a service formal description can be obtained.

For example, given a microservice description F and features f_1 ("outputting a set of resources") and f_2 ("handling images"), some feature definitions can be the following:

$$f_1 \in F \Rightarrow |Output| > 1 \quad (1)$$

$$f_1 \in F \wedge f_2 \in F \Rightarrow \forall x(x \in Output \rightarrow image(x)) \quad (2)$$

Feature definition 1 formalizes feature f_1 by stating that the service's output cardinality has to be higher than one. Meanwhile, feature definition 2 formalizes that all output resources are images, and is applicable for services that (i) output a set of resources (as of feature f_1) and (ii) handle images (feature f_2). Therefore, the formal description of a service that is feature-oriented-described by f_1 and f_2 would be:

$$(|Output| > 1) \wedge \forall x(x \in Output \rightarrow image(x)) \quad (3)$$

It can be observed that feature definition 1 is activated by the presence of one feature in the feature-oriented description. Therefore, that definition serves to formalize one feature. However, feature definition 2 is activated by the presence of two features in the feature-oriented description.

Allowing definitions that are activated upon the presence of more than one feature might be regarded as unnecessary complexity. However, this versatility is justified. Consider feature f_3 as "The service stores a resource that is provided as input". When using f_2 with f_1, a postcondition should be set (i.e. "*multiple images will be returned*"), but when used with feature f_3, a precondition should be set (i.e. "*an image has to be provided as input*"). This can be achieved by setting one definition for f_2 and f_1 (as shown in definition 2), and another definition for f_2 and f_3, as shown next:

$$f_2 \in F \wedge f_3 \in F \Rightarrow \forall x(x \in Input \rightarrow image(x)) \quad (4)$$

This serves to resolve the issue of feature interaction, already identified in feature-oriented programming (Prehofer, 1997). By describing features semantically, feature descriptions can be combined to produce a semantic service description.

Microservices framework has a vocabulary of terms that can be extended, with each term representing a feature. As the framework follows the REST architectural style, terms for HTTP GET, POST, PUT or DELETE requests are already part of the vocabulary.

As a feature-oriented description framework, the microservice approach has the following advantages: (i) it allows the reuse of feature descriptions among different services, and (ii) it reduces the description task to selecting a set of features that describes the considered service, given a vocabulary of terms.

3 A VOCABULARY FOR SEARCH MICROSERVICES

In this paper, we define a vocabulary for describing of services that perform retrieval operations, i.e. search services. In order to narrow the modelling task, we will only consider image search services. These

kinds of services attempt to fulfil the user's goal of finding a particular kind of image.

In order to perform this task, we will follow the next steps:

1. Identify popular image search services.
2. Collect features across the considered services.
3. Model features through feature definitions.
4. Adapt services to fit the modelled interface.

3.1 Service modelling

We will consider Google Images, Panoramio, Bing Images and Flickr search as search services that will be analyzed and modelled. Many more image search services are available, but we have based our decision on the fact that:

1. Flickr offers capabilities that are not present on the other services, such as tag search.
2. Panoramio offers additional capabilities that are not present on the rest of services, such as search by location.
3. Google Images and Bing Images have similar capabilities, but offer a different interface. This fact could impact on the vocabulary.

After identifying the services that are considered for the modelling task, we will collect raw features from these services. By observing each of these services, we can see that some features define the obvious behaviour of a search service and are enabled by default, while others require additional configuration in the service, often accessible through an "advanced search" option. Therefore, for this task, we accessed "advanced search" and help pages of the services in order to discover their capabilities.

It is immediate to identify raw features such as "the service returns a set of images as search result". However, this raw feature can be split into finer-grained features, such as "returning a set of results", "returning an image" and "performing a retrieval operation". The combination of these three features would produce the firstly identified raw feature. By splitting raw features into finer-grained features, more feature combinations are possible.

Table 1 summarizes the features that we have identified in the considered services. The first column shows raw features that were identified in the services, while the next column represents the specific finer-grain features that are produced. The third column is a shorthand term that represents the feature and, finally, the last four columns indicate the presence of these features in each service. As anticipated, Google Images and Bing Images share the same capabilities, while Flickr and Panoramio services serve to enrich the vocabulary in order to cover a wider set of features.

The next step is modelling the features, so that a service description can be produced out of a list of features. In Microservices framework, this is done by defining feature definitions, i.e., sets of logical preconditions and postconditions that are activated on the presence of a specific set of features. An example is the fact of outputting images: feature `image` and feature `get` should set the postcondition that the service's output has to be an image, which, more formally, is:

$$\texttt{image} \wedge \texttt{get} \Rightarrow \forall x(x \in \mathit{Output} \rightarrow \mathit{image}(x))$$

This feature definition solves the feature interaction between `image` and `get`.

Therefore, for each of the identified features, a set of feature definitions have been built that consider the possible feature interactions. The identified feature definitions are shown in table 2. The first column shows the features that are involved in each definition, and the next two columns show the preconditions and postconditions. Microservices framework uses Javascript notation to express conditions, but we have used Mathematical notation for this purpose. Also, although Microservices framework also allows defining a textual description for each feature definition that enables documenting of the service, we have omitted these textual descriptions in table 2 for clarity purposes.

These feature definitions allow producing a formal service description, defined by a set of preconditions and postconditions. This service description can be processed to perform operations such as automatic validation (validating preconditions and postconditions), automatic discovery of the semantics of inputs, or automatic user interface generation, as will be seen in section 4.

Microservices framework allow wrapping existing services into services that fit a specific interface. In our case, we have defined a vocabulary of terms that allow describing, at least, the considered services. However, in our model we have made assumptions such as resource attributes' names or input parameters' names. In order for the services to fit this interface, an adaptation has to be performed. For each service, we have defined an adapter, i.e. a piece of code that adapts input parameters, and a data extractor, i.e. a collection of XML selectors that transforms HTML's unstructured information into structured data. This lets using the existing services with the interface that has been defined in the vocabulary.

Raw feature	Feature	Term	Google	Bing	Flickr	Panoramio
A set of images is returned to the user as search result	Retrieval operation performed	`get`	✓	✓	✓	✓
	Multiple resources involved	`multiple`	✓	✓	✓	✓
	Service deals with images	`image`	✓	✓	✓	✓
Images are filtered according to a location	Resource is filtered by location	`location-filtered`	✗	✗	✗	✓
Images are indexed from anywhere in the web	Resource cannot belong to another domain	`local`	✗	✗	✓	✓
Images are summarized and have a thumbnail	Resource is summarized	`summarized`	✓	✓	✓	✓
	Resource has a title	`titled`	✗	✗	✓	✓
Images can be searched by keywords	Resource is filtered by given keywords	`keyword-filtered`	✓	✓	✓	✗
Images can be searched by tags	Resource is filtered by given tags	`tag-filtered`	✗	✗	✓	✓
Images can be searched by dimensions	Resource is filtered by dimensions	`size-filtered`	✓	✓	✓	✗
Images can be searched by creation or taking date	Resource is filtered by creation date	`creation-filtered`	✗	✗	✓	✗
	Resource is filtered by publishing date	`publishing-filtered`	✗	✗	✓	✗
Images can be selected by license	Resource is filtered by license	`license-filtered`	✗	✗	✓	✗
Images can be searched by colour and content	Resource is filtered by colour	`colour-filtered`	✓	✓	✗	✗
	Resource is filtered by content	`content-filtered`	✓	✓	✗	✗

Table 1: Analysis of detailed features in the considered services

3.2 Discussion

The considered services have been adapted for the defined interface and are available online [1]. Each service can be executed through a user interface that is generated automatically by analysing the services' feature definitions. Also, a documentation of the service can be obtained by combining the feature definitions. Additionally, when executing the service adapters, the preconditions and postconditions are checked, which allows automatic validation of the services. Also, microservices descriptions are published as Linked Data, as well as the services' output.

Some additional aspects of the approach are worth mentioning. For example, both Bing and Google Images allow filtering images by content. Bing allows image filtering by photographies, illustrations and content filtering by face or face and shoulders. Instead, Google supports 'clip art' filtering, as well as photos, illustrations and face filtering, but not face and shoulders filtering. Therefore, although both of them have the capability of filtering images by content type, the specific feature is slightly different in each of them. In order to reduce complexity, we will assume that both services share the `content-filtered` feature, and adapt the input appropriately at the services' adapters. This means, however, that the specific feature of this filtering capability for each service cannot be discovered from the service description.

Finally, the feature-oriented approach allows reusing features across different services, even in the case that they belong to different domains. Many of the modelled features can be combined to describe services that do not target image retrieval, but resource retrieval in general. A video search service would only need to introduce a new feature ("dealing with videos") and define a few feature definitions. Similarly, a document search service is simply a subset of the defined vocabulary (i.e. to define a web page search service, the `image` feature should be left out). Storage services (i.e. those performed with the HTTP POST and PUT methods) would require more feature definitions, as most definitions interact with `get` feature.

4 CASE STUDY: METASEARCH SERVICE

A metasearch service[2] has been implemented as a case study for the defined vocabulary. This metasearcher aggregates results from the described

[1] `http://lab.gsi.dit.upm.es/microservices/proxies`

[2] `http://lab.gsi.dit.upm.es/microservices/metasearch`

Features	Preconditions	Postconditions
`get`	$method(service) = get$	$status(service) = 500$
`image get`	-	$\forall x(x \in Output \rightarrow image(x))$
`multiple get`	-	$\lvert Output \rvert > 1$
`summarized get`	-	$\forall x(x \in Output \rightarrow \exists y(uri(x,y)))$
`keyword-filtered get`	$\exists i(i \in Input \wedge name(i,\text{“}keywords\text{”}))$	$\forall x(x \in Output \rightarrow keywords(x) \subseteq i)$
`tag-filtered get`	$\exists i(i \in Input \wedge name(i,\text{“}tags\text{”}))$	$\forall x(x \in Output \rightarrow tags(x) \subseteq i)$
`local get`	-	$\forall x(x \in Output \rightarrow domain(x) = domain(service))$
`titled get`	-	$\forall x(x \in Output \rightarrow \exists y(title(x,y))$
`size-filtered image get`	$\exists i(name(i,\text{“}size\text{”}) \wedge i \in Input \cap \{small, medium, big\})$	$\forall x(x \in Output \rightarrow size(x,i))$
`creation-filtered get`	$\exists i(i \in Input \cap Dates \wedge name(i,\text{“}creation\text{”}))$	$\forall x(x \in Output \rightarrow creation(x) < i)$
`publishing-filtered get`	$\exists i(i \in Input \cap Dates \wedge name(i,\text{“}publication\text{”}))$	$\forall x(x \in Output \rightarrow publication(x) < i)$
`license-filtered get`	$\exists i(name(i,\text{“}license\text{”}) \wedge i \in Input \cap \{attribution, modifiable, commercial\})$	$\forall x(x \in Output \rightarrow license(x,i))$
`location-filtered get`	$\exists lat, lng, r(lat, lng, r \in Input \cap \mathbb{R})$	$\forall x(x \in Output \rightarrow \lvert\lvert x - (lat, lng) \rvert\rvert < r)$
`content-filtered get`	$\exists i(name(i,\text{“}content\text{”}) \wedge i \in Input \wedge i \subseteq \{face, shoulders, clipart, illustration\})$	$\forall x(x \in Output \rightarrow content(x,i))$
`colour-filtered get`	$\exists i(name(i,\text{“}colour\text{”}) \wedge i \in Input \cap \{bw, r, g, b\})$	$\forall x(x \in Output \rightarrow color(x,i))$

Table 2: Specification of features

services (i.e. Google Images, Bing Images, Flickr and Panoramio). As these services have heterogeneous features, the problem is not trivial.

The metasearcher works as follows. First, it considers a microservice description that includes all the features shown in table 1. Then, by analizing the conditions that result from combining feature definitions, it identifies required inputs and their types. With this set of inputs, it builds a user interface that allows executing the service.

However, as can be observed in table 1, no service has all the mentioned features. The metasearcher will then select matching services according to the features that are used in each query. In order to do so, the metasearcher validates preconditions for each feature. If the precondition is not valid, the feature is considered inactive. Only active features will be considered when filtering services. Then, the matching services are executed and their results are aggregated.

For example, let's consider that a user interacts with the metasearcher by providing some keywords, picking an image size, and clicking the submit button. Then, `location-filtered`, `tag-filtered`, `creation-filtered`, `publishing-filtered`, `license-filtered`, `coloured-filtered` and `content-filtered` features will be deactivated, as long as their preconditions are not satisfied (as long as their required inputs are not provided).

It is remarkable that the process of detecting the features to deactivate is not immediate. An insatisfied condition belongs to a feature definition, which can involve many features. For example, if no keywords are passed to the metasearcher, it has to decide whether to deactivate feature `get` or feature `keyword-filtered`, as both are involved in the definition that requires keywords as input. We consider that the number of feature interactions is an indicator of the importance of a feature. Therefore, the employed criteria has been to perform the minimum number of deactivations of definitions (and thus the minimum number of deactivation of feature interactions), and an algorithm has been implemented for this purpose.

Also, the activation state of certain features cannot be identified automatically. This is the case of the `local` feature and `titled` feature, which are present only in Flickr and Panoramio, and do not set preconditions. These features are thus not considered by the metasearcher for service matching, as their inclusion would make that only Flickr and Panoramio are able to match the required features.

The metasearcher has been implemented as a generic microservice aggregator, in which a set of features are selected for service matching and aggregation. This means that this implementation is independent of the considered features, as the user interface is built by analyzing a microservice description, and the features are selected according to the satisfaction of their preconditions. We have also experimented on aggregating plain search services with same satis-

factory results, allowing to aggregate image results as well as document results. This makes this implementation an interesting foundation for the definition of an abstract microservice aggregator based on sets of required features, which however is out of the scope of this paper.

5 RELATED WORK

In our paper, we have addressed the issue of describing search services using lightweight semantics. WADL (Hadley, 2006) defines a format for building and publishing RESTful semantic service descriptions which can be discovered and processed by automatic agentes. Other similar RESTful approaches are SA-REST (Sheth et al., 2007) and hRESTS (Wright State University, 2008), which allow building semantic service descriptions by annotating textual descriptions of services' APIs with RDFa and Microformats (Microformats community, 2008), respectively. Also, RDForms (Baker, 2005) attempts to "add to the Semantic Web capabilities similar to HTML forms". In this approach, schemas for indexable, container and settable operations are defined, which represent HTTP GET, POST, and PUT methods, respectively. All these approaches focus on facilitating service description, but do not consider reuse of features across different services, as they are not feature-oriented. Also, a vocabulary for the description of search services in these approaches is missing.

On the specific topic os search services, OpenSearch (A9.com, inc., 2005) is an approach to the description of search services. By creating an OpenSearch description document, a search service is published. The description document enumerates the services' capabilities and configurations. OpenSearch delegates to service providers to build an appropriate RESTful interface that can be described using OpenSearch. Some services might require adaptation in order to be described by an OpenSearch description, but this issue is not addressed. Finally, as OpenSearch is limited to search services, it considers fine-grain aspects such as content encoding. In our case, these details are missing, as complexity is moved to service adaptation on behalf of a simpler description framework.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have reviewed the current state of the art in semantic service description and discussed how access to Linked Data could be improved by automating the execution of search services.

We have used Microservices framework to build a vocabulary that can be employed to build semantic service descriptions for image search services in the web. The solution is a small set of terms that illustrates the feature-oriented approach of the framework and highlights the versatility of lightweight semantics. The feature-oriented approach has provided a solution that generalizes well to other search services while it is specific to the targete domain. The vocabulary has been used to semantically describe a set of services that are aggregated on a metasearcher as a case study.

Future works regard automatic creation of user interfaces and services, and automatic detection of features through pre and postconditions. Preliminary approaches have been shown in the case study, where appropriate user interface controls are shown for each search option, based on the semantics of each input parameter, and where appropriate feature sets are selected according to the provided inputs.

ACKNOWLEDGEMENTS

This research project is funded by the European Commission under the R&D project ROMULUS (FP7-ICT-2007-1) and by the Spanish Government under the R&D project *Java sobre Ruedas* (FIT-350401-2007-8).

REFERENCES

A9.com, inc. (2005). OpenSearch specification. `http://www.opensearch.org/Specifications/OpenSearch/1.1`.

Adida, B. and Birbeck, M. (2008). RDFa Primer - Bridging the Human and Data Webs. `http://www.w3.org/TR/xhtml-rdfa-primer/`.

Apel, S., Leich, T., Rosenmller, M., and Saake, G. (2005). Combining feature-oriented and aspect-oriented programming to support software evolution. In *In AMSE05, at ECOOP05*.

Apel, S., Leich, T., and Saake, G. (2006). Aspectual mixin layers: aspects and features in concert. In *Proceedings of the 28th international conference on Software engineering*, page 131. ACM.

Baker, M. (2005). RDF Forms. `http://www.markbaker.ca/2003/05/RDF-Forms/`.

Berners-Lee, T. (2006). Linked data. *Retrieved April*, 12:2008.

Bracha, G. and Cook, W. (1990). Mixin-based inheritance. In *Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*, pages 303–311. ACM New York, NY, USA.

Chakrabarti, S., Van den Berg, M., and Dom, B. (1999). Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640.

Elrad, T., Filman, R. E., and Bader, A. (2001). Aspect-oriented programming: Introduction. *Commun. ACM*, 44(10):29–32.

Erl, T. (2005). *Service-oriented architecture: concepts, technology, and design*. Prentice Hall PTR Upper Saddle River, NJ, USA.

Fernández-Villamor, J. I., Iglesias, C., and Garijo, M. (2010). Microservices: Lightweight service descriptions for REST architectural style. In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence*.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California.

Hadley, M. J. (2006). Web application description language. `https://wadl.dev.java.net/wadl20061109.pdf`.

Harrison, W. and Ossher, H. (1993). Subject-oriented programming: a critique of pure objects. *ACM Sigplan Notices*, 28(10):411–428.

Lopez-Herrejon, R. (2005). Understanding feature modularity in feature oriented programming and its implications to aspect oriented programming. In *ECOOP2005 PhDOOS Workshop and Doctoral Symposium, Glasgow, Scotland*.

McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic Web Services. IEEE Intelligent Systems.

Microformats community (2008). Microformats. `http://microformats.org/`.

Murphy, M., Dick, M., Fischer, T., Fraunhofer, I., and Stuttgart, G. (2008). Towards the" Semantic Grid": A state of the art survey of Semantic Web services and their applicability to collaborative design, engineering, and procurement. *Communications of the IIMA*, 8(3):11–24.

Prehofer, C. (1997). Feature-oriented programming: A fresh look at objects. *Lecture Notes in Computer Science*, 1241:419–443.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web Service Modeling Ontology, Applied Ontology. IOS Press.

Sheth, A. P., Gomadam, K., and Lathem, J. (2007). SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. In *IEEE Computer Society*.

Steimann, F. (2000). On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1):83–106.

Trujillo, S., Batory, D., and Diaz, O. (2007). Feature oriented model driven development: A case study for portlets. In *Proceedings of the 29th international conference on Software Engineering*, pages 44–53. IEEE Computer Society.

Vitvar, T., Kopecky, J., and Fensel, D. (2007). Wsmo-lite: Lightweight semantic descriptions for services on the web. In *Proceedings of the Fifth European Conference on Web Services*, pages 77–86. Citeseer.

Wilde, E. and Gaedke, M. (2008). Web Engineering Revisited. In *Proceedings of the 2008 British Computer Society (BCS) Conference on Visions of Computer Science, London, UK (September 2008)*.

World Wide Web Consortium (2004). OWL-S: Semantic Markup for Web Services. `http://www.w3.org/Submission/OWL-S/`.

Wright State University (2008). HTML Microformat for Describing RESTful Web Services and APIs. `http://knoesis.wright.edu/research/srl/projects/hRESTs/#hRESTs`.

Zhou, J., Koivisto, J.-P., and Niemela, E. (2006). A survey on semantic web services and a case study. In *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pages 1–7.