Realizing Service-Finder

Web Service Discovery at Web Scale

Emanuele Della Valle, Dario Cerizza, Irene Celino and Andrea Turati
CEFRIEL – Politecnico di Milano, Milano, Italy
firstname.lastname@cefriel.it

Holger Lausen and Nathalie Steinmetz
seekda
Innsbruck, Austria
firstname.lastname@seekda.com

Michael Erdmann and Wolfgang Schoch
Ontoprise
Karlsruhe, Germany
erdmann@ontoprise.de

Adam Funk
The University of Sheffield
Sheffield, United Kingdom
a.funk@dcs.shef.ac.uk

***Abstract*— The Web is moving from a collection of static documents to one of Web Services. Search engines provide fast and easy access to existing Web pages, however up till now no comprehensive solution exists which provides a similar easy and scalable access to Web Services. The European research project Service-Finder is developing a platform for service discovery where service related information from heterogeneous sources is automatically integrated in a coherent semantic model to allow effective discovery**

Web Services, Automatic Annotation, Semantic Search, Web 2.0

I. INTRODUCTION

Initially the Universal Description Discovery and Integration (known more often by its short name as UDDI [1]) was proposed as a solution to publish and search for public Web Services, but it did not prevailed in the domain of publicly available Web Services. UDDI defines data structures relevant to service discovery and a set of APIs to access them. The standard was supported by major soft-ware vendors such as IBM, Microsoft and SAP. Although UDDI was claimed to become successful in restricted and controlled environments and successful implementations of UDDI servers exists, none of the initial public UDDI registries exists any more.

In spring 2007, we identified several drawbacks and pitfalls of public UDDI registries and we submitted a proposal for a European Project named Service-Finder. The proposal was funded and the project started in January 2008. In its first nine months of activity Service-Finder designed and implemented a prototype that has already managed to overcome most of identified problems.

In this paper we present drawbacks and pitfalls of public UDDI registries (Section II) and how we believe is possible to overcome them (Section III). In Section IV and V we respectively present use cases for Service-Finder in the form of story boards and the requirements we derive from them. The architecture of Service-Finder and its components are presented in Section VII. Section VIII provides some insight view of physical deployment of Service-Finder. Finally, we described the progress we are making beyond state-of-the-art (Section IX) and we conclude discussing the potential impact of Service-Finder (Section X).

II. PITFALLS OF PUBLIC UDDI REGISTRIES

Service Oriented Architectures (SOAs) along with Web Services technologies are widely seen as the most promising fundament for realizing service interchange in business to business settings. However, it is envisioned that SOAs and Web Services will increasingly move out of these settings and out onto the Web. Several previous research projects (just to mention SWWS, DIP, ASG, InfraWebs or NeP4B) proved that descriptions on a semantic level (e.g., WSMO [2]) can be used to enable precise discovery of services. The critical problem in this new Web oriented environment is one of scale because services appear, disappear and change at a rate much higher than in business to business settings. Current SOAs and Web Services technologies require large amounts of human effort related to finding services, choosing the most relevant, integrating selected services within new applications, and enabling interoperation between them. One of the essential building blocks for creating applications that utilize the vast quantities of services, which are available on the Web is making it easier to discovery and select the right services.

Current technologies do not sufficiently address the description and matchmaking of requests from service consumers with the available service offerings of providers. UDDI was initially proposed as a component of Web Services usage process enabling registering and discovering services, but finally UDDI did not reach its expected potential. We have identified four main reasons, why UDDI did not prevail in the domain of public Web Services:

- UDDI is centered around programmatic access to the registry and only a few mostly technically focused user interfaces are available. There is a considerable learning curve required for somebody not familiar with Web Services to understand the technicalities of UDDI specification. If we use an analogy to Web of pages, all what the new user of Web of pages is required to do to start exploring websites, is a pre-installed browser. It is important that early adopters of Web Services technology, who learns about it for the first time, should be able to start exploring it with a few simply steps.
- There are no policies such as availability checks for Web Services in place and thus many services in public registries have been outdated. The value of the service in the public UDDI registry is minimal if the service itself does not exist anymore. Web is unpredictable and services can appear and disappear (the same as websites), but there must be a mechanism allowing to eliminate these services which are not available any more. Popular search engines eliminate

websites from their indexes, which are not working for some period of time.

- There are no means for community feedback. Most of Web Services published on the Web are far from being perfect and community feedback is one of elements, which can enable their improvement. The UDDI specification basically provides no support for community features. Practically there is only one possibility to provide feedback allowing the user to contact a provider by email listed in the service description. But in time of Web 2.0 application, this minimal support for contact between service provider and service consumer is not enough. While email might work for some particular cases, it does sustain a community around providers of Web Services so in the long run the UDDI model has little applicability to the Web.
- It is hard to expect that users would select a service based just on its WSDL definition and a short description. To make decision about applicability of the service, service consumer need to become familiar with pricing, terms and condition, service level agreements to name just a few. UDDI as API centric approach neglects that using a Web Service means first understanding it and making a contract this requires learning and understanding various aspects of the service, not just its interface and short description.

III. OVERCOMING UDDI LIMITATION

Today, when the time of public UDDI registries is practically over, the process of publishing Web Services is simply done by providing an access to the interface description (i.e. WSDL) and providing a couple of Web pages explaining the peculiarities of a service. Popular examples of such approach are provided by Amazon[1] and eBay[2], but a large number of less popular company are following their example.

This information is then found by the crawlers of search engines such as Google or Yahoo. Via keyword search potential service consumers can find those pages and from there also the technical specification to integrate it with their applications. In addition to using standard search engines a couple of portals (e.g. XMethods[3], WebServiceX.NET[4], Web Service List[5], StrikeIron[6], RemoteMethods[7], eSynaps[8]) are available that are specialized on providing access to public services. However most of them only cover a small portion of the actually available services and suffer from the problem of missing or outdated information [3].

seekda[9], which is one of the partner of Service-Finder consortium, currently has managed to crawling almost 28.000 Web Services from more than 7000 Web Services providers. However, crawling, indexing and making WSDL files searchable is not enough; the support for regularly verification of all crawled services as well as community features are needed to overcome the limitation of UDDI.

[1] http:// aws.amazon.com/
[2] http://developer.ebay.com/
[3] http://www.xmethods.net/ve2/
[4] http://www.webservicex.net/WCF/
[5] http://www.webservicelist.com/
[6] http://www.strikeiron.com/
[7] http://www.remotemethods.com/
[8] http://www.esynaps.com/
[9] http://www.seekda.com

The aim of the Service-Finder project is to create an architecture and implementation that significantly improves the current situation by:

- Employing automated methods to gather WSDL and all related resources such as a wiki that documents the operations, a blog in which developers are discussing any webpage in which useful information about the Web Service are give.
- Leveraging semi-automatic means to create semantic service descriptions of a Web Service out of the information available on the Web.
- Facilitating community feedback to build up and improve semantic annotations automatically.
- Describing the aggregated information in semantic models and allow reasoning over them.
- Providing a Web 2.0 portal to support users in searching and browsing for Web Services, to give recommendations to users and to track user behavior improving accuracy of service search and user recommendations

IV. USE CASES FOR SERVICE-FINDER

In contrast to UDDI that try to discover services within restricted areas, e.g. within given service repositories, Service-Finder aims at addressing publicly available services, i.e. publisher and consumer do not know of each other prior to the discovery process and there is only little control over the format and quality of the service annotations. Hereafter we illustrate examples of such use cases in the form of story boards.

A. Use Case 1: System Administrator

This Use Case describes a system administrator (Sam Adams) that is looking for an SMS Messaging service that he could build in into his application.

Sam Adams is a system administrator at a bank. He is responsible for the banks online payment facilities that need to be online and working 24 hours a day, 7 days a week. In case of a system failure, Sam needs to be immediately alerted to be able to the problem as soon as possible. Therefore Sam would like to use an SMS Messaging service that sends him an SMS to alert him in any case of emergency.

Sam Adams is searching for a specific service on the Service-Finder portal, an SMS service that he can use within his application to get alerted. His goal is to find a service given his specific constraints. We can divide his constraints into crucial ones and negligible ones. His crucial constraints are the immediate service delivery and the reliability of the service (service non-availability of at most 1 hour per year). To Sam Adams more or less negligible constraints belongs the pricing model. He would prefer to pay a low base fee (preferably none) and transaction fees, as he expects that he will only get up to 50 SMS per month.

Sam Adams knows the Service-Finder portal and knows that he can find lots of useful services there. Now he uses the portal to find a specific service. He knows exactly what he is seeking for and has carefully specified his needs. He will search for appropriate services and will then focus on the service details to figure out which one fulfills his goals best.

Sam visits the Service-Finder portal. He wants to search for a specific service, an SMS Messaging service that he can use to be

alerted by SMS in urgent cases. He enters the keywords “SMS Alert” into the search box and launches the search.

Sam gets a list with matching services. He would like to choose how many resulting services he can see on one page. Also he would like to see some condensed/summarizing information with each service on the result list. Ideally he would expect to get a short overview of what functionality each service is providing, as well as who is the provider and what is the service availability (measured availability). He could use such information to decide whether he wants to look at a service in more detail or not.

Sam first checks the short explanations of the services on the first result page. He expects to find the most relevant services for his search topic on the first pages. Now he needs to look whether the delivered services are providing the functionality he is looking for, i.e. sending SMS messages. Therefore he wants to be able to look, for each service, at more detailed information. These could be information on the service features as pricing model, availability (both measured and promised by the provider), provider information (as e.g. contact information), service coverage, etc., as well as on user feedback (i.e. community input) and links to related information on the Web.

After looking at the details of some services, Sam realizes that the services he got as result to his search term provide quite different functionalities and belong to different service categories, some offering SMS messaging services, others offering services that can be used to alert users on certain occasions, but not via SMS (e.g. via voice messaging). Thus he would like to directly see the different categories to which the services belong and then choose to see all services belonging to the category that is interesting him most (e.g. 'SMS Messaging'). If possible, he would like to further filter his search by browsing through categories this way (e.g. seeing next all categories that the SMS Messaging services belong to, if the categories are hierarchically defined).

After Sam has refined his search in the above described way, he knows that all services he has in the result list now are valuable for him in the sense that they all provide an SMS messaging service. Now he would like to look for services that are provided by an Austrian provider and that have no base fees.

Now Sam sees all SMS messaging services that are provided by Austrian providers and that have no base fees. He realizes that there is only a very small number of providers and decides to deselect the country facet. Now he sees all services that have no base fee, regardless of where the provider is from. Now he needs to choose services that come into question for him, i.e. services that provide a very high reliability. Therefore he would like to sort the services according to their actually provided availability (as measured by the Service-Finder portal).

Sam now sees the results ranked by their availability, which is the most crucial selection criterion for him. Next he would like to compare the measured availability of a service to the availability promised by the service provider and to the availability actually provided, according to the service provider. He does so by looking at each service's details. At the service details he also looks at the service coverage, to see whether the service is actually covering Austria, where Sam lives and works.

He would then like to compare different services according to their transaction fees. The user rating that a service got from the community is also an interesting point that Sam would like to compare for different services. Looking at the details of a few services Sam realizes that it is quite hard to compare the different services this way. He would like to be able to see the search results in a more structured way that allows him to directly compare the services, according to their provider, their availability, their fees, their user rating, etc. Ideally Sam would like to be able to select some services from the intermediate search results list, e.g. the services with the highest availability that he then would like to compare in a table view.

As last step, Sam would like to try out the service and thus check whether the service is actually doing what it should. But of course this depends on whether the service provider actually offers free service trials or not. If so, the portal should display this, so that Sam can use the portal's Web Service Tester to test the service functionality.

B. Use Case 2: Business User

This Use Case describes a business user (Bud Hughes) uses the Service-Finder portal in order to find a service that he can include into his project.

Hughes is a Business and Technology Consultant and his main activity is the translation of diverse business processes and associated requirements into a business solution. He is at the moment working on a project in the e-health domain that aims at developing an application that should help doctors in getting a second opinion for a specific clinical case.

The scenario he wants to deal with can be depicted as follows. A patient goes to his/her General Practitioner (GP), who needs a second opinion. For that reason, the GP uses the application - which is the one Bud is thinking about - to require an advice. The GP sends the patient's exams and the application has to forward it to a Specialist. The Specialist recognizes a particular disease and writes the report, which the application has to send back to the GP via fax. At the end, the application has to support the GP in the payment of the Specialist's advice.

In order to realize whether this application is easily feasible or not, Bud wants to exploit Service-Finder to find the required services. The services he needs are: (i) a service that allows a user to send a file with a very big size via e-mail - which is used to send a patient's exams along with other useful information -, (ii) a service that sends faxes - which is used for sending the final report to a GP -, and (iii) a service that manages payments. Initially, Bud focuses on the service for sending large files via e-mail.

Bud knows that he can find many services on the Service-Finder portal and now wants to find a specific service. He therefore focuses on the description of the details of every service returned by a search.

At the Service-Finder home page, Bud writes “send email” in the search box. He examines the results: the first service is named “ValidateEmail” and Bud discards it, the second one is named “fax” so Bud rejects it, the third one is named “SendSMSWorld” and it is not what Bud is looking for. Soon he realizes that he needs a sort of filter, because the results include many services that perform very different actions. In particular, he would like to select a category name and/or some technical features.

Anyway, Bud notices that the fourth service returned by the search is named “Communication” but its abstract says “Service to send large files via e-mail by uploading files to the zeta server and generating an e-mail message with the appropriate download hyperlink”. Therefore, Bud selects it in order to view the description of its details.

First of all, he would like to read a description of the functionality provided by the selected service. Since he is a business user, he would be happy to read detailed information about the provider such as its rating, information about its customers and so on. Moreover, he would like to have some guarantees such as a service level agreement and the pricing model. After reading all those information, he really appreciates the service characteristics and he wonders if the service provider offers services that are related to the one he is examining or if another alternative service exists or if any service is recommended to be used along with the current one.

Bud is enthusiastic because the first service he selected satisfies all requirements of the project he is working on. He is looking forward to sharing his discovery with his colleagues involved in the same project. For that reason, he would like to send them a link to the description of that service, but he would prefer to send different links: a link to the more technical details for the “developer team” and a link to the more “business” details to the project supervisor.

Bud thinks that the service he selected is fine; however his thoughtful nature pushes him to investigate other services. Before leaving this service description he would like to save it in a sort of bookmark basket, in order to be able to come back to it at any moment.

Moreover, Bud would like to access at every moment the history of the searches he performed, in order to view any variation in the result set.

Bud leaves the service description and he would like to come back to the search results without pressing the “back” button of his browser many times.

After reading the generic description of another service, Bud wants to compare it with the service that for now is the best one in Bud's opinion. He does not want to compare long descriptions belonging to different pages manually; he would prefer to have a sort of table that resumes the most important characteristics of the selected services side by side.

From the comparison table Bud realizes that also the second service satisfies his requirements and is cheaper too. However its provider's description contains only few details and no reference that Bud can use to contact him. Bud is suspicious and thus wants to collect opinions from those users that used the service. In particular, he would like to view a summary of the ratings that users gave it overtime.

Bud is happy to see that the second service, the cheaper one, is very well rated by the other users. So he decides to discard the first service he found and to exploit this service. He would like to enter into a contract with the service provider to regularly use it.

After, Bud goes on in order to find the other two services he needs.

C. *Use Case 3: post-editing service information*

This Use Case describes a web developer (Will Daniels) that, after using a service listed on Service-Finder, decides to edit the information on the portal in order to improve it for other community users.

Will Daniels was already familiar with the Service-Finder portal, He did actually find an appropriate SMS service and did figure out that he can use it for authentication by sending a token via SMS. Using this service he can avoid double registrations and ensure high quality of the user data.

With this in mind Will feels that the categorization found on the Service-Finder portal (Services > Communication > SMS) is not sufficient and would like to add the SMS service to the category "Authentication". If this category does not yet exist, he would like to add it now manually. If he cannot add it directly, he would like to contact the portal administrator and make a request to add the category. If he needs to be registered to the portal to be able to do changes, Will logs in now.

The information about the SMS service he has found contains a reference to a terms and condition document of the provider web page, however Will found the actual terms of the SMS service. He wants to update the description to include the more specific URL describing terms and conditions, in order for himself to find it again, but also to help others that may be interested in the same service. He uses the “correct this” button he sees next to every information item and enters the new URL.

Will is actually using the SMS service now, and has discovered some discrepancies between its description on the Service-Finder portal and his experiences using it. Using his bookmark he quickly finds the service's entry again on the portal. Will now edits the user-added tags and the wiki description that other portal users have made for the service in question.

Will would like to have a look at the edit history of the tags and the wiki information in order to see if maybe one same user made several previous mistaken (in Will's opinion) tags or wiki entries.

After Will did the corrections on the user-added information, he wishes to alert the portal administrators to report the incorrect tags and wiki information. He expects the administrators to eventually restrict the rights of users that attract attention due to their bad feedback. Furthermore, if Will did eventually not have the right to change user-added information by himself, he expects the administrator to do so after he reports the problems.

Concerning the SMS service that Will is currently using he did not only find some wrong information, but he also thought that some important information was missing. Now he thinks that it would be useful to share this information with the other users belonging to the community. For that reason, he would like to improve the “wiki pages” associated to that service explaining how to use the service exactly.

Will found the SMS service by searching for services that offer free trials, however he figured out that this particular service only offers a free trial after performing registration at the service provider. Other services he found like the stock quote service of xignite2 did offer free trials also without registration. While he was happy to do sign-up with the SMS provider, he was unhappy about the lack of information about this difference on the Service-Finder portal, i.e. concerning the portal-provided generic service information. Since Will uses the portal often, he wants to improve it and add “requires sign-up” to the “offers free trial” property.

V. SERVICE FINDER REQUIREMENTS

In the following we will summarize the requirements on the Service-Finder system that we have identified within the previous Use Cases (see Section III). We have grouped similar requirements together into three main categories: Search related, Web Service information related and User Community related.

A. *Service-Finder Web Service Search Interface*

SEARCH - The portal must provide search functionality.

PAGINATION - The portal must provide a configurable pagination of the search results.

RANKING - The portal should rank the services in the result lists according to a given metric, like e.g. the relevance of the search term.

SORTING - The portal must provide functionality to sort search results according to user's selections.

GET-CATEGORIES - The portal must provide the categories a service is belonging to and allow the user to view all services belonging to a specific category.

BROWSE - The portal must allow the user to browse through categories.

FACETED-SEARCH - The portal should provide a faceted search functionality that allows to select either one facet or to combine more facets.

COMPARISON-TABLE - The portal should provide a side-by-side comparison table for services.

SELECT-COMPARISON - The portal should provide the user the possibility to select services that he wants to compare.

DISAMBIGUATION - The portal may provide a "did-you-mean" functionality by displaying a separate box with categories that match a keyword request. This can as well help to solve disambiguations, if for example a keyword can be used with different meanings in different domains.

SHORT-OVERVIEW - The portal must provide some short overview information for each service in the result list.

VIEW-DETAILS - The portal must provide a possibility to view all details of a service.

GROUP DETAILS - The system should group the service details on the basis of their type (generic features and technological features) and allow a user to view only a subset of the service details at a time

BOOKMARKS - The portal should provide the possibility to bookmark single services and put them in a sort of basket.

SEARCH-HISTORY - The portal may provide the possibility to at any moment see the whole history of searches one so far did on the portal. This would e.g. allow the user to sort of store its searches, so that he can easily re-do the same query simply by clicking on it.

TESTER - The portal should provide a Web Service Tester, i.e. a try-out functionality.

B. *Web Service Related Information*

SERVICE-INFORMATION-MUST - The description of a Web Service must include (1) a service interface, i.e. an URL on the Web, and (2) a service provider.

SERVICE-INFORMATION-SHOULD - The description of a Web Service should include information as: (1) how can the service be used, (2) its payment modalities, (3) its terms and clauses, (4) user-added information as ratings, comments and tags, (5) measured values of service levels such as availability (uptime) or performance (response time) and (6) the declared SLAs. The user ratings information should provide a summary of the ratings that users gave to a service recently, by subdividing them on different periods (e.g. the ratings collected during the last month, during the last 6 months and during the last 12 months).

SERVICE-INFORMATION-MAY - The description of a Web Service may include links to other services, either (1) from the same service provider, or (2) belonging to the same service category.

PROVIDER-INFORMATION-MUST - The description of a service provider must include the name of the provider and its references (for contact purposes).

PROVIDER-INFORMATION-SHOULD - The description of a service provider should include the rating of the provider (independently from its services).

PROVIDER-INFORMATION-MAY - The description of a service provider may include advanced information, such as the quantity of the provider's customers as well as some indication of its size and the type of its business. Such information may not result from analyzing Web pages but may be added by users or directly by the providers.

C. *Service-Finder User Community Interface*

RATING - The portal must allow a user to rate a service.

COMMENT - The portal should allow a user to write a little comment to a service, e.g. to justify his vote.

TAGGING - The portal must allow a user to tag a service.

WIKI - The portal should have a “wiki” section in which users can describe services collaboratively.

ASSIGN-CATEGORY - The portal must allow the user to manually assign services to categories.

ADD-CATEGORY - The portal should allow the user to manually add new categories to the portal or to make a respective request at the portal's administrators.

EDIT-GENERIC-INFO - The portal should allow the user to edit (e.g. correct) generic information belonging to a service.

ADD-GENERIC-PROPERTIES - The portal should allow the user to extend the existing generic properties used to describe the service and provider properties.

ADMIN-EDIT-RIGHTS - The portal must allow the Service-Finder portal administrators to edit, i.e. for example revert and override user's entries (tags, wiki entries).

EDIT-ADDED-INFO - The portal should allow any user to manually edit already previously added tags and wiki entries or to make a respective request to the portal administrators.

EDIT-HISTORY - The portal should maintain a history of all user-added edits (e.g. wiki entries, tags)

CONTACT-ADMIN - The portal should allow the user to contact the administrator and tell him eventual problems he detected on the Service-Finder portal.

RESTRICT-USERS - The portal should allow the Service-Finder portal administrators to restrict specific users' accounts if they suspect misuse or incompetence.

SEND-LINK - The portal should allow a user to “send a link” of a service description to persons with different skills (e.g. developer or manager), and to add explanatory information to the link, providing the possibility to add information like e.g. why a person receives that link,

who the sender is, the category of the service, its abstract and its average rating

REGISTRATION - The portal must offer user registration for some advanced features. It must make clear the advantages of being a registered member of the portal, and thus belonging to the Service-Finder community, as a user would accept to register himself only after having had a successful experience using the portal and after understanding all the advantages and privileges of portal members.

RECOMMENDATIONS - The portal should provide a user with recommendations, in two different cases: (1) when providing the description of a Web Service to a user, the system should recommend other services selected by other users along with the current one, or (2), exploiting a given user profile; the system should suggest the user potentially useful services.

VI. Requirements for External Components

The Service-Finder portal will provide an efficient access to publicly available services by gathering the Web Services, as well as related information, from the Web, creating semantic annotations of the services, using mostly this information found on the Web.

To be able to gather as much as possible relevant service related information, the Service-Finder project is depending on external components, as e.g. WSDL descriptions, external Web sites containing service descriptions, service price lists, etc. (mostly, but not necessarily, provider-related), as well as Wikis and Blogs containing user feedback and rankings of services.

A. *Provider-Related Requirements*

Service crawling begins with the assumption that a publicly available Web Service has somewhere on the Web a corresponding interface description published. We are focusing at start on those described by the WSDL.

Having identified those domains that host Web Services we start a focused crawl in order to obtain as much related information as possible. This leads us to the requirements that the Service-Finder has for the provider Web sites.

First of all it is important that they provide at all information, verifiable over an URL, about the services they offer. This information should be located on the same domain than the service description itself, to allow a service crawler to find the information when starting from e.g. a WSDL description location. The related information can include e.g. information about the service coverage, about the service availability, FAQs, etc. Price lists are another important factor. The service providers should offer detailed plans concerning their pricing modalities.

The targeted Service-Finder portal will provide a Web Service Tester functionality. This will allow eventual service users to invoke and test the service to check whether it delivers the anticipated functionality. To be able to offer this functionality, we are dependent on free service trials offered by the service providers, at least concerning all commercial, non-free, services.

B. *Service-Related Information Requirements*

Service-Finder try to gather as much as possible service related information on Wikis, Blogs, Web pages, etc., where service users describe their experiences with given services.

Service descriptions may exist in different versions on the Web. These need to be correlated among themselves and with any related information that is available about them, be it provider related or not.

C. *Requirements Related to Other Service Standards*

As described above, we will at the beginning of the project focus on WSDL service descriptions. These are both easier to detect on the Web and easier to analyze, as they have a standardized interface. Nevertheless not all publicly available services are described in WSDL, providers often use e.g. the REST paradigm or JASon. Thus, in a second step we will try to define methods to detect other service descriptions than WSDL on the Web.

VII. Architecture and Components

We design Service-Finder portal and its whole architecture in order to include a deep understanding of all relevant aspects of the portal. The architecture we describe in this section includes automatic gathering of Web Services at global scale, high precision in both searching and ranking, as well as the building of an easy- to-use Web 2.0 inspired user-interface.

To obtain a common understanding of the Web Service domain we are first developing two major ontologies: A generic Service-Finder Ontology, which contains all aspects related to services, providers and users that are relevant to the different components. Although not all information that is stored within the Service-Finder architecture will be physically stored in the ontology repository, all of the data items are described in this ontology. The second ontology is the Service Category Ontology. This ontology provides a separation of the overall domain of public services into various domains, such as messaging and finance.

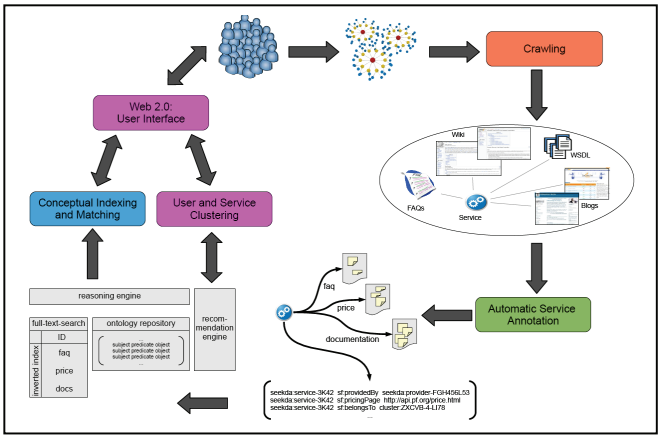

Figure 1. Overview of Service-Finder components and data flow

To obtain a common understanding of the Web Service domain we are first developing two major ontologies: A generic Service-Finder Ontology, which contains all aspects related to services, providers and users that are relevant to the different components. Although not all information that is stored within the Service-Finder architecture will be physically stored in the ontology repository, all of the data items are described in this ontology. The second ontology is the Service Category Ontology. This ontology provides a separation of the overall domain of public services into various domains, such as messaging and finance.

Fig. 1 gives a high level overview of the components and the data flow between them. The overall cycle starts with a Web developer publishing a Web Service.

A crawling component harvests the Web for the interface descriptions describing the actual services (WSDL documents). The crawler identifies services. A service is not necessarily corresponding to a single document: an interface description might include several services or, more common, several versions of interface descriptions describe the same service. Once a service is detected the crawler starts also to look for any related information. This related information can be on the same domain than the service description as well as on other domains. For commercial services the provider typically has webpages detailing terms and conditions, pricing model, general documentation etc. The output of the crawler is a structured and focused snapshot of the Web, which maps services to one or more interface descriptions as well as zero or more related webpages.

This snapshot of the Web is then analyzed by the Automatic Annotation component. This component further structures the information related to a service and extracts semantic statements conforming to the Service-Finder ontologies. One task for example is to categorize the services according to the Service Category Ontology, i.e. to determine whether a service is providing financial information or is performing operations in the bioinformatics domain. The crawler, in the first step, uses very fast algorithms when trying to determine if a particular document is relevant or not. For example whether a document is relevant or not will be initially determined by the Web link graph. The annotation component will then further analyze the documents and for example discard directory listings and other auto generated pages. Moreover pages will be classified into their genres, such as “FAQ”, “pricing”, “user comments”, etc. All information will be stored as triples conforming to the Service-Finder ontologies. In addition the relevant subset of the Web documents will be packaged to allow full text search by subsequent components.

The conceptual indexer and matchmaker stores all extracted information. It combines full text keyword search on the unstructured part of the index, such as generic textual documentation with ontological querying and reasoning. For example a rule might express that when a service has a pricing page and some documentation pages hosted on the domain of the provider it is a commercial service. A user could then combine a keyword query with an ontological query such as “give me all commercial SMS services in the US”.

A Web 2.0 style user interface will display data to the user. Besides displaying data, the interface will also allow to modify, delete or add semantic metadata to the services. For example if a user has found a particular service that is relevant to him, he can add a rating. Or if the user interface displays a particular URL as “pricing page” and the user discovers that the content is not related to the service he can correct this by deleting this particular statement. In this way the users can correct the information extracted by the automatic annotation component. The feedback will be used to further improve the extraction algorithms.

In addition to the explicit feedback Service-Finder will also gather implicit feedback hidden in the behavior of the users. By analyzing the click streams recorded by the Service-Finder portal we will build up user and service clusters. Those clusters can then be used as input for a recommendation engine that can suggest suitable services for a given user context.

A. *Ontologies*

In this section we briefly describe the ontologies that manifest the shared understanding of the domain. We are initially developing the ontology schema using the NEON-Toolkit[10]. After the initial version has matured to some extent, we intend to further develop it using the semantic media Wiki[11]. From the ontology language point of view we are not assuming any advanced features such as class definitions by universal or existential restrictions, but mainly explicit class hierarchies and properties. Therefore the exchange format of the ontology will by RDF(S).

In Section IV.B we have identified the data objects that are required to satisfy the user’s needs to find Web Services on the Web. First we identified that users like to have some kind of categorization scheme that helps them to orient and navigate within the existing set of services, these can be very generic categories, like research, communication etc. Secondly there are a number of properties of a service that is of particular interest. These properties include provider, availability, documentation, service-level-agreement and so on. Existing ontologies for services focus on the functional description of services (OWL-S [4] and WSMO [2]). They provide a framework for describing the functionality; however they do not include any detail on the more pragmatic properties. SA-WSDL [5] is a more light weight approach that builds directly on top of WSDL. It explicitly supports categorization, however defers the development of a categorization schemes to the end user.

The categorization scheme and the generic service ontology can be then orthogonal to the existing efforts. The Service-Finder ontologies are specific to the use case of publicly available services. In other scenarios it might be of less importance who the provider of a service is or where to find suitable documentation.

1) *Generic Service-Finder Ontology*

The generic Service-Finder Ontology will be primarily developed and maintained as part of the conceptual indexer and matchmaker. This component stores all the structured and unstructured information obtained by crawling, analyze and user feedback. We model also those entities that might not be directly stored within an ontology repository, such as the history information of a community descriptions or clustering information about services and users.

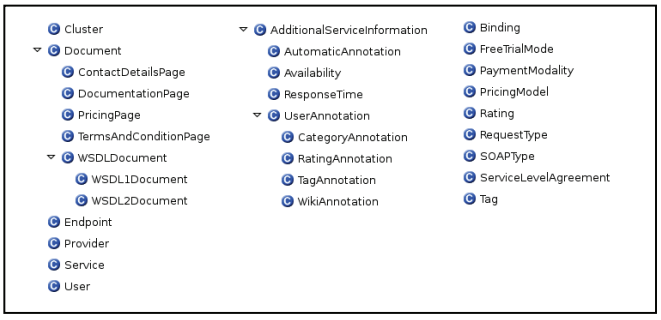

Figure 2. Overview of the generic service ontology

Fig. 2 shows an overview of the main concepts. The central entity is a service that is associated with several other entities. Several “pieces” of Additional Information can be gathered around a service. This additional information can stem from the user community as well as from the automatic annotator, respectively the Web. Information extracted by the annotator is obtained by analyzing several documents that can be related to a service. In addition to the service and the information and documents connected to it, we model the

[10] http://www.neon-toolkit.org/

[11] http:// semantic-mediawiki.org/

provider as the entity that operates a specific service, as well as the user that searches and eventually uses services.

2) Service Categories

The purpose of the Service Category Ontology is to provide a coarse grained classification to allow users to get an overview of available services. We provide a set of categories based on the experiences gathered during the initial services crawling and operation of the service search-engine at seekda.com. Thus we believe that the chosen categories cover the major domains in which currently publicly accessible services are available.

Providing a categorization is rather difficult. Pure labels are ambiguous and interpreted differently by different individuals. On the other hand when a user wants to explore the space of available information, he does not want to spend time on understanding a complex set of descriptions. We have looked at the various categorization schemes used by existing portals such as esigma[12], remotemethods[13], strikeiron[14], webservilist[15], wsfinder[16], wsgoogle[17] and programmableweb[18]. Fig. 3 depicts the two top levels of the resulting ontology proposal.

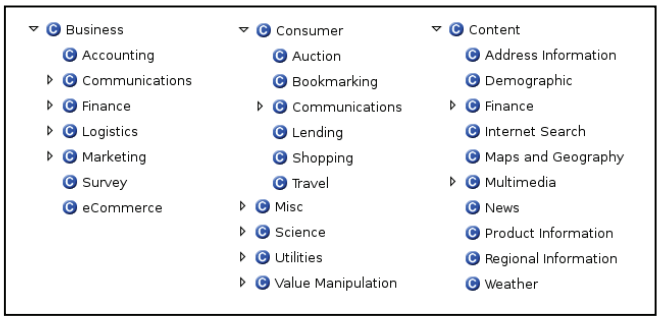

Figure 3. Overview of the category ontology

We will describe the category of Content services in more detail by giving a small set of samples for part of the categories.

Address information These are services that validate, correct or complete address information.

- http://seekda.com/providers/serviceobjects.net/DOTSAddressPlus - The DOTS Address Validation Plus Web Service is a programmable Web Service hosted by Service Objects and used by businesses to integrate address correction, validation, and enhancement services into their software applications and web sites.
- http://seekda.com/providers/strikeiron.com/GlobalAddressVerification - The StrikeIron Global Address Verification Web Service instantly validates and enhances addresses from over 240.
- http://seekda.com/providers/addressdoctor.com/FastCompletion - FastCompletion service for address data, e.g. for data entry support in a call center, will generate complete addresses, for instance from a postal code and the beginning of the street name

Demographics These services provide demographic information about a specific region or user group. Information can be crime rate, house hold income, etc.

- http://seekda.com/providers/serviceobjects.com/DOTSGeoCoderCanada - DOTS Demographics is an XML-based Web Service that allows developers to append demographics information to any home listing, customer, or prospect in the country. This service is used to examine the demographics of your market area.
- http://seekda.com/providers/strikeiron.com/CensusData - To use this Web Service, simply enter a ZIP, city, county or state and receive a detailed profile containing economic and housing characteristics for the given input. The data can then be used within any application, platform, product or Web site.
- http://seekda.com/providers/srlink.com/ChannelAnalysisService - Levelsoft BusDirectory ChannelAnalysisService Web Service provides up-to-date retail business distribution analysis using geographic, demographic, social and economic statistical information, compiled and verified from various governmental and controlled bodies.

Finance These services retrieve financial information, for examples stock quotes.

- http://seekda.com/providers/xignite.com/XigniteQuotes This Web Service provides multiple quote related operations including several quote formats (simple, extended), market summary information, and top market movers, losers, and gainers.
- http://seekda.com/providers/webservicex.com/StockQuote Get Stock quote for a company symbol by using this Web Service
- http://seekda.com/providers/cdyne.com/DelayedStockQuote CDYNE PowerQuote data is obtained from official Wall Street sources and comes complete with their "100 Availability" SLA commitment.

Internet Search These services provide a functionality to programmatically search the Web (or pieces thereof).

- http://seekda.com/providers/amazonaws.com/AlexaWebSearch - The Alexa Web Search Web Service offers programmatic access to Alexa's Web search engine. The Alexa search engine is a Web-wide search engine that powers the Web search on the Alexa website.
- http://seekda.com/providers/g-vo.org/RASSConeSearchesService - A new product, the German Astrophysical Virtual Observatory, offers a cone-search on the photon event file, containing all the photons observed in the RASS. The same user interface offers query capabilities. The size of this catalogue, which contains about 100 million photons, puts more requirements on the tuning and indexing of this database.

B. Service Crawler

The Service Crawler produces a snapshot of the relevant part of the Web with appropriate metadata to allow efficient analyze.

[12] http://esigma.com/telos/discover/browseservices.html/
[13] http://remotemethods.com/
[14] http://strikeiron.com/servicecategories.aspx
[15] http://webservicelist.com/
[16] http://wsfinder.jot.com/ WikiHome
[17] http://wsoogle.com/
[18] http://www.programmableweb.com/

1) Web Snapshot

Being unfeasible to run the extraction algorithm envisioned for the annotation component on all available Web resources given the limited resources of this project, the task of the crawler is to deliver a snapshot of the Web that contains only the relevant part of the Web. To ease the processing in subsequent components this snapshot should allow easy access to resources related to a service or provider.

The only input required for the component is a set of seed URLs. Seed URLs will be provided by domain experts and based on the crawl already performed for the seekda.com portal.

The component produces consistent snapshots of the relevant part of the Web in periodic intervals such as every 6-8 weeks up to 5GB compressed data (200.000 documents). In order to deal with a large number of files we adopt the Internet Archives Arc File Format[19]. To make it easier for the Automatic Annotation component to access this bunch of data the crawler also produces indexes: one allowing random access to any resource crawled given its URL, another one containing the list of all known service IDs together with their related WSDL files, and a final one containing the list of all known service IDs together with a list of resources that are deemed related (and information on why).

2) Identifing Services

It is a well known problem that a particular page in the Web is sometimes duplicated several times and accessible using different URLs[20]. The problem of duplicated content is not only relevant for regular Web content, but also for Web Services. Therefore, a central task of the crawler is to identify services. Our first experiments indicate that only one third of all well formed WSDLs retrieved do represent meaningful unique services. We determine a unique service ID by concatenating the name of a service definition with effective top level domain of the endpoint(s) listed in the WSDL file.

3) Identifing Related Information

Once services are identified a crucial aspect is to gather as much meaningful information as possible to aid the task of discovery. Therefore it is the task of the crawler to not only harvest interface descriptions (WSDL files), but also any Web resource that are potentially related. We consider the Web link graph to determine which Web resources are related. Resources linking to or linked from the WSDL file can be considered as "related".

C. Automatic Service Annotator

The annotation component adds semantic metadata to the services based on the information extracted from the service interface descriptions and the related Web resources. First the component associates every service with a category; second it analyzes the related information, discards irrelevant documents and as well determines the genre of each document (e.g. price information, terms and conditions, etc.). One service can be associated with more than one category. For example if a service validates mobile phone numbers and also offers to send text messages (SMS) it would be in the category "messaging" as well as "data validation". If possible it extracts additional statements like "service x offers free trials after sign-up". Finally all information is exported as ontology adhering to the generic service ontology, respectively to the service category ontology.

[19] http://www.archive.org/web/researcher/ArcFileFormat.php

[20] http://googlewebmastercentral.blogspot.com/2006/12/deftly-dealing-with-duplicate-content.html

1) Categorizing Services

The annotation component has to associate one or more categories to every service. The component can do this based on the WSDL file(s) associated with the service and taking into account the related information. In section VII.A.2 we have presented the initial draft for the service category ontology and a set of samples associated with some of the categories. In addition to the data given in the respective section, we use the seekda tagging feature to generate an initial learning set. We adopt the convention to prefix the category with a colon, such that services within a category can be retrieved via the URL (e.g.http://seekda.com/services/tags/:<categoryname>).

2) Document Relevance

The annotation component receives as input a list of services together with the list of documents that it deemed related. However the crawler has only limited capabilities to process documents. The annotation component must re-assess if a given document is indeed relevant with respect to a particular service. It must annotate the documents deemed to be relevant with the meta information why it is relevant and with respect to which entity (service ID or provider ID).

3) Genre Detection

Related information is different in its nature. It can be anything from a pure directory listing that links to a couple of services to a marketing text describing the advantages of a particular service. All different genres are described within the generic service ontology. For providers we generally can find a contact page and a home page. For services related pages can be directory listings, price listings, FAQs, comments, usage guides, general documentation, terms and condition page and so on. Once the annotation component has extracted the genre it has to add the respective annotation, i.e. the genre and why a particular genre is assumed.

4) Document and Keyword Corpus

Once document relevance and genre detection is performed the component may discard certain documents previously (by the crawler) deemed relevant. The annotator produces a keyword corpus for the conceptual Indexer and Matchmaker. The keyword corpus should contain one virtual "document" for each entity. This document has multiple "sections" with text / keywords stemming from multiple sources. For example it should contain sections for "name", "operations", "input and outputs", "description", etc. This document is intended to be used as base for the full text search.

5) Additional Information

In addition to the extraction/annotation described above that is mainly on a document level the component should also extract knowledge about the service or provider directly. The items to be extracted are listed in Section V.B and are detailed as concept and attributes in the generic service ontology. As one example we can take the general detection of the pricing model. It will be very difficult to extract concrete prices such as $100 setup, $20 base $0.10 per message, etc. However for the end user it is already important to know whether a service has a pricing policy at all, or is just free to use, or requires a sign-up and attribution policy.

D. Conceptual Indexer and Matcher

The conceptual indexer and matcher is the central data store for all information that has to be used by multiple components within the Service-Finder architecture. It provides semantic querying capabilities on top of the data stored that allow to do matchmaking between user request and service offers as well as retrieval capabilities for example

to render a service page or to retrieve user feedback on automatically extracted annotations.

1) Indexing

The component stores the semantic annotations from the annotation component as well as from the user interface. While the annotation component will typically deliver multiple statements at once (e.g. all annotation for one service), the user interface performs updates in a smaller granularity. When storing the annotations the component should maintain the source of the information, i.e. it should be able to separate between facts created (or retracted) by users and those stored by the annotation component. As additional annotation seekda will provide a ranking information associated with every service. The component also stores the data obtained by the clustering component.

The indexer also stores and indexes the textual description provided by the annotation component, as well as the textual comments provided by users.

2) Querying

The component allows querying the semantic data available in the repository. Queries can originate from the user interface in order to render provider or service details as well as from the annotation component for example to obtain user corrections of annotations.

In order to allow the user to intuitively create queries the component allows combining a keyword search with an ontological query. Such a query can for example be: give me all “sms" services provided by xyz.com.

E. Cluster Engine

The cluster engine uses the implicit and explicit feedback that users of the Service-Finder portal leave when they interact with the portal in order to derive clusters of users and services. It does do by identifying from users' history, those users that behave similarly and, for each group of users, by identifying the services they usually interact with and group services used by users belonging to the same cluster.

It takes as input service annotation data (extracted + user feedback) and click streams from user interface to extract user actions. Such information include: detecting the friends of each user, the explicit preferences (those manually inserted by the user during registration, or edited after registration), the service annotation, the keyword written in the search box, the disambiguation actions, the request for sorting results according to a property, the request to view details of a given service, the time interval spent by the user to read a service description, the request to try-out a service, the browsing through categories and tags, the request to compare two services, the action of bookmarking, rating, commenting, tagging and categorizing a service, the request to add a category, and the action of adding a property to a service description.

The component finds (unlabeled) clusters of users and services. Such clusters will be used to recommend services to users. Such information is used by the Conceptual Indexer And Matcher for service recommendation, that is identifying a set of services related to a given service for a specific user.

F. Service-Finder Interface

Service Finder Interface represents the main entry point for a user who wants to search for services. It provides the users with search functionalities to help them in finding the most appropriate services tofulfill their needs.

In particular the user can:

- search services by keyword, tag or concept in the categorization;
- sort and filter query results by refining the query;
- compare and bookmark services for those services that offer this functionality, try out the service;
- register to the SF Portal and contribute in a Web 2.0 fashion by tagging, rating, commenting and adding descriptions/properties to services; and
- allow developers to invoke Service-Finder functionalities through an API access to service data

The user interface queries the Conceptual Indexer and Matcher and its data structures in order to retrieve: ordered list of services matching a query, details about a service as provided by the whole Service-Finder system, details about a service as provided by the community of Service-Finder users, details about a set of services (in case of comparison), details about a service provider, service category ontology and the most used tags provided by the users. Moreover it queries current seekda system in order to get: information about service availability and access to try out functionality for those services that provide it.

1) Search Interface

The approach to search interface is manifold: users are offered with the possibility of a standard keyword search, but also with a controlled search; tag-based search and concept-based search (with concepts from the Service category Ontology) belong to this second type. Since, at search time, the system is “aware” of those controlled folksonomies and taxonomies, the user can also be presented with a disambiguation functionality (as explained in [6] and [7]): once the user enters a keyword, the system proposes possible meanings of his request by suggesting tags or concepts that can be related to the inserted keywords.

Our past experience in designing, implementing and evaluating Squiggle (see http://squiggle.cefriel.it and [6]) proves that this disambiguation is a highly appreciated feature in a search engine, which helps the “beginner” users to familiarize with innovative engines by re-conducting their syntactic searches to the respective semantic ones.

2) Navigation Interface

Once the user has submitted his/her initial query, the system retrieves a list of services that match the request. Therefore, the user is presented with a set of results to be scanned to find the one(s) which best fit his needs.

In this situation, a simple plain list containing a high number of results can appear as confusing to the user. The Navigation Interface adopts techniques and approaches to further help user navigation across search results.

Apart from the disambiguation feature described above, we investigated the applicability of faceted browsing/search to search results: the user is presented with a number of “facets” which help him to: restrict his query: by selecting a value in a facet the result space is limited to those services matching the introduced constraint; expand his query: by generalizing his request to a broader “meaning” (e.g., by selecting a super-concept of a selected concept); change/move his query: by adding/removing constraints to the current query.

Moreover, as described in [8], we aim at employing semantic navigation principles to help users to browse across the result space following ”meaningful” paths. Our past experience in designing, implementing and evaluating the STAR:chart framework (see http://seip.cefriel.it) based on the STAR:dust meta-model (see [9]) proves the value added by faceted browsing and semantic navigation fundamentals.

3) *Web 2.0 Features*

The advent of Web 2.0 wave and the so-called “wisdom of the crowds” brought a new awareness in Web surfers with regards to their attitude towards content generation on the Web. The idea that the Web consumer is often also a provider of content (thus becoming a prosumer) is the basis of the Web 2.0 “revolution” and led to the rising of new interaction patterns mediated by a new family of tools, of which wikis represent the most common, widespread and well-known example.

In designing the Interface, we took into due consideration this trend, by enriching the search engine with collaboration capabilities. The users, therefore, are provided with the possibility to add ratings, comments, tags, annotations and generic descriptions/contributions to service details; this “wiki-way” to user editing can be highly improved (as described, among others, in [10]) by the use of the semantic technologies employed in the Service-Finder project.

The data provided by the users, together with their “behavior” in searching and navigating across results, will in turn serve as basis to the elaborations within the Cluster Engine.

VIII. PHYSICAL DEPLOYMENT

In this section we depict the physical deployment of the various software components necessary to realize Service-Finder. The amount of data and processing required exceeds the capacity limit of a single machine.

For the crawling we can expect to process about 40 pages per second (including link extraction and relevance rating), that is close to 100 million pages per month per machine. A study [11] of 2005 reports that Google claims to have indexed more than 8 billion Web pages; that means in 2005 it would require 80 machines to only crawl the available data. However since we pursue only focused crawling, we will not need to crawl the whole Web, but only a part of and expect a high coverage using 2-3 machines.

When analyzing documents more thoroughly to obtain semantic annotations more processing time is required. Maynard et al. [12] state that for example named entity recognition can deal with about 10kb of text per second. Assuming 10 documents related to every service and a document having 10kb of pure textual data that would mean that on a single machine the data corresponding to 30,000 services would need a bit more than 3 days only for entity recognition.

Finally the textual and semantic data needs to be stored and indexed. If we consider the performance of a standard full text search engine like lucene[21] we can index about 20 documents per second, and we can assume that a single machine can answer about 5 retrieval requests a second with a medium sized repository (about 1 million documents). In addition to indexing the text also the semantic data must be loaded / indexed. Of course the figures heavily depend on many configuration parameters such as memory, stopword list, parser configuration etc. However it gives a rough indication that with the initial data set we will not need a dedicated machine to build the indexes.

The front end server renders the pages and transforms the data from the indexer and matchmaker into a nicely human readable format. The amount of resources required depend heavily on the number of users of the portal and on the technology used to implement the user interface. However given that a single machine easily handles a view for dozen concurrent users this should suffice for the initial deployment.

Finally the clustering is envisioned as an offline process since it potentially requires a large amount of computation.

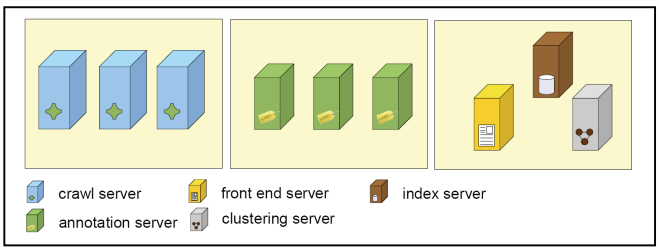

Figure 4. Physical architecture

Fig. 4 depicts the initial physical deployment for Service-Finder. Machines are grouped. Every group resides in the same network segment such that data transfer is relatively fast. Different groups of machines might reside in different network segments, such that some optimization should be applied when transferring data. The data to be exchanged between machines can be summarized as:

- crawler → annotator: every 6-8 weeks consistent snapshot of relevant part of the Web.
- index → annotator: copy of internal database every week synchronized
- annotator → index: after an entity, e.g. service, all its related annotations are updated
- user interface → clustering: log files are shipped in intervals on a daily or weekly base, as required.
- clustering → index: after completion of cluster algorithm new information is sent

IX. BEYOND STATE-OF-THE-ART

One of the major limitations of the solutions designed so far is that they require services descriptions have been published in a dedicated repository before the discovery process can be executed. One of the projects that aim at avoiding the advertisement of services is named Woogle. Woogle is a search engine that provides a Google-like interface by which a user can perform keyword-based queries. Woogle does not require that one publish Web Services into the system, because it obtains Web Services directly from several UDDI registration nodes. Obviously, in this way the problem remains, because the Web Services have to be published into UDDI. For this reason, now, an efficient infrastructure for service discovery that does not expect the advertisement of services descriptions is missing. Service-Finder aims at providing such an infrastructure.

Currently, Web Service repositories offer little and poor quality metadata and the main obstacle in improving the situation lies with

[21] http://lucene.apache.org/java/docs/benchmarks.html

asking humans to provide, and continuously update, the required metadata, which is high cost. In addition, semantic service annotation requires a domain ontology from which to draw relevant concepts and relations. However, again, the creation and maintenance of such ontologies is an expensive task.

Consequently, the service annotation work in Service-Finder focuses on automating the process of building semantic service descriptions for already published Web Services. It will build on recent advances in the area of learning Web Service domain ontologies and its application to the problem of semantic annotation of Web Services.

The European Project WonderWeb[22] pioneered the use of ontology learning tools in the context of Web Services by investigating and (partially) answering some fundamental questions. The main finding was that the textual sources have a low grammatical quality and that they use language in a specific way (i.e. they employ a sublanguage). These characteristics constrain the choice of approaches that can be used. In addition, the concepts that should be extracted, should reflect both static (domain concepts) and procedural (functionalities that Web Services offer) aspects of the domain. More recently, ongoing work in the European Project TAO[23] is addressing some limitations of these early approaches by using unsupervised data mining techniques on the source code and available documentation and then combining the knowledge learnt from these sources into a domain ontology, so that the service provider can be assisted in the creation of the service descriptions. At the same time, the manually created rule-base methods from WonderWeb have been tested on several Web Service repositories, for the purpose of automatic creation of service descriptions from documentation and WSDL files.

The semantic annotation work in Service-Finder goes beyond this state-of-the-art in a number of ways. Firstly, we will address automatic creation of service descriptions for a much wider range of already existing Web Services and specifically address the issue of continuously updating the metadata, as the service API changes. Secondly, Service-Finder utilizes information from Web 2.0 sources, especially blog postings, and will not require the availability of the source code itself. Thirdly, Service-Finder offers a shared Web-based user interface where the automatically created metadata can be post-edited and enhanced further by the user community.

Below, we provide an overview of how and to what extent the project will go beyond the state of the art describing the main scientific and technological innovations. At the end, we provide a tabular summary (i.e. a feature-by-feature comparison).

The Service Crawler is achieving progress beyond state-of-the-art in realizing focused crawlers for Web services. A particular hard problem is to assess whether a given Web resource contains information about some service (i.e. related information) or not. Moreover, it has to be investigated how far the automated detection of services using the WSDL standard can be extended to services using other standards such as REST or JASON.

The Automatic Service Annotator is advancing the state of the art in large-scale metadata annotation of existing Web services, applied to a comprehensive Web service repository. New algorithms have been developed and compared against previous work, using existing gold standards and developing new ones to incorporate relevant Web 2.0 sources, such as blog posts. A particular novel dimension is dealing with contradictory information extracted from different pages and using a community process to verify the automatically created metadata.

The Conceptual Indexer and Matcher is advancing the state of the art by integrating formal knowledge (service meta-data, ontologies) and informal (textual) knowledge into one query-mechanism for Web services. The formal part of the service finding algorithm will be extended to also cover non-structural knowledge which is more complex such as context information or policies to enhance precision and recall of the query results compared with current, less expressive approaches.

The Cluster Engine is developing new algorithms for grouping users into clusters based on the similarity of their behaviors, so that the limitation of simple clustering techniques based on user's profiles are overcome. In addition, using statistics and graph theory, the Cluster Engine is advancing the state of the art by investigating how to cluster services by taking into account user clusters. Indeed, several generic clustering techniques exist, but none of them is able to address the problem of clustering services and users contemporaneously, in the sense that an alteration on user clusters impacts on service clusters. Moreover, it is investigating how these clusters can be exploited for both performing optimizations and customizations and refining service descriptions.

In the Interface efforts we are focusing on some aspects that have been neglected so far in realizing interfaces for service discovery: (i) exploring ways and means of integrating semantic data and Web 2.0 features, (ii) determining how to monitor user actions and which actions to be monitored, (iii) customizing the interface according to user's profile, as well as (iv) investigating the possible advantages that user profiles can lead to in term of human-machine interaction.

To wrap up, the key research innovation of Service-Finder are centered on deriving a comprehensive set of information about publically available service by complementing advance automatic service annotation techniques with user/service clustering ones. In particular the Automatic Service Annotator creates Web Service descriptions by analyzing WSDL and related information coping with contradicting information and using community process to verify results. Whereas, the Clustering Engine investigates and implements techniques for: clustering users accordingly to their behaviors and clustering services accordingly to their usage by users belonging to the same clusters. Moreover in the Conceptual Indexer and Matcher we are carrying on several innovative engineering activities in order to be able to apply semantic technologies not only to semantic annotation generated by the Automatic Service Annotator, but also from the users that interact with the Web 2.0 features of the Interface.

X. CONCLUSIONS

Service-Finder is developing means to enable the discovery of Web Services at a Web scale. Similar to how current search engine have enhanced classical information retrieval techniques by scaling them to the size of the Web and by using the Webs internal structure (links) as improvement.

Service-Finder targets at core aspects of the Future Internet that is the converged communication and service infrastructure that will gradually replace the current Internet, mobile, fixed and audiovisual networks. As a matter of fact, such a convergence is only possible if more and more information and services are provided in a way tangible to machine to machine collaboration instead of solely providing it via mark-up languages like HTML and transfer protocols

[22] http://wonderweb.man.ac.uk/

[23] http://www.tao-project.eu/

like HTTP. While those techniques are in principle developed, their use is still in its infancy due to the lack of support for various steps of the process in facilitating services. Service-Finder demonstrates how to enhance current technologies to enable efficient discovery of services.

When looking closer at the current service composer and mash-up portals[24], they provide an impressive functionality by enabling end users without much technical knowledge to build composed application out of basic building blocks. However having a closer look, they all suffer under the same problem: a lack of underlying usable services. In fact, this is not caused by a lack of their existence, however services are either not published using the right protocols and descriptions or they are – but due to the lack of an efficient discovery platform, they cannot be found. Service-Finder is a research and demonstration project providing its contribution exactly in this space.

Service-Finder actions provide core mechanisms required to allow ICT systems to adopt within changing environments:

- Using Web principles such as openness, robustness and existing metrics within published content;
- Explicit and Implicit User Interaction as integral task for construction, improvement and validation of rich service description; and
- Semantic Web technologies as means to organize internally the data on available services.

The combination of above cornerstones enables the location of services, thereby we increases the efficiency of service provision and subsequently enable service usage with the ambitious but achievable goal of making the service-oriented-architecture scale to millions of services on the Web.

Service-Finder contributes to the overall goal of the Future Internet by providing means to make existing services accessible and discoverable. We simplify the service publishing process by removing the burden of any registration and bring service selection to the end-user by providing an intuitive interface to the search engine allowing also non-technical persons to assess the quality of a service.

- Creators become able to design more communicative forms of content by integrating third party services in their own creation process.
- Publishers increase their productivity, by being able to provide complex services without the need to register explicitly them with discovery or brokerage services.
- Organizations can automate their processes by quickly finding adequate services that can fulfil particular needs within a more complex process.

Finally, even if Service-Finder is conceived for the open Web, the achievements of the Service-Finder are applicable in different scenarios such as:

- Application integration scenarios in large enterprises, especially companies engaged in some kind of eCommerce, which will actively use Service-Finder to look for “address validation”, “fraud detection”, “credit worthiness”, etc.
- Service-Finder as Appliance. Once the project has demonstrated the feasibility of our approach, it could be developed further such that it could be packaged to an Intranet-Appliance for large organisation. The Service-Finder appliance would provide services search capabilities at very low installation and maintenance costs, due to the automated crawling and annotation.

[24] Teqlo, Yahoo! Pipes, Openkapow

ACKNOWLEDGMENT

The work described in this paper is has been partially supported by the European project Service-Finder (FP7- 215876).

REFERENCES

[1] OASIS UDDI Technical Committee: The UDDI Technical White Paper. Technical report, OASIS (2004)

[2] Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services { The Web Service Modeling Ontology. Springer (2006).

[3] Daniel Bachlechner, Katharina Siorpaes, Holger Lausen, and Dieter Fensel. Web service discovery a reality check. In 3rd European Semantic Web Conference, 2006.

[4] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic markup for web services. W3C Member Submission, available from http://www.w3.org/Submission/2004/07/, November 2004.

[5] Joel Farrell and Holger Lausen. Semantic annotations for wsdl and xml schema. http://www.w3.org/TR/sawsdl/l, August 2007.Fsdfs

[6] Irene Celino, Emanuele Della Valle, Dario Cerizza, and Andrea Turati. Squiggle: An Experience in Model-Driven Development of Real-World Semantic Search Engines. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, ICWE, volume 4607 of Lecture Notes in Computer Science, pages 485-490. Springer, 2007.

[7] Irene Celino, Emanuele Della Valle, Dario Cerizza, and Andrea Turati. Squiggle: an application framework for model-driven development of real-world Semantic Search Engines. In W3C SWD SKOS Use Cases and Requirements Material, http://www.w3.org/2006/07/SWD/wiki/EucSquiggleDetailed, 2006.

[8] Irene Celino and Emanuele Della Valle. Multiple Vehicles for a Semantic Navigation Across Hyper-environments. In A. Gómez Pérez and J. Euzenat, editors, ESWC, volume 3532 of Lecture Notes in Computer Science, pages 423-438. Springer, 2005

[9] Irene Celino, Emanuele Della Valle, and Francesco Corcoglioniti. The STAR:dust conceptual model: Semantic Travel Across Resources (STAR) with the aim of designing unified support tools (dust) to help travelers in their navigation. In W3C SWD SKOS Use Cases and Requirements Material, http://www.w3.org/2006/07/SWD/wiki/EucStarDustDetailed, 2006.

[10] Irene Celino, Francesco Corcoglioniti, and Emanuele Della Valle. Towards a Semantic Contact Management. In Anna V. Zhdanova, Lyndon J. B. Nixon, Malgorzata Mochol, and John G. Breslin, editors, FEWS, volume 290 of CEUR Workshop Proceedings, pages 64-77. CEUR-WS.org, 2007.

[11] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web, pages 902-903, New York, NY, USA, 2005. ACM.

[12] D. Maynard, V. Tablan, C. Ursu, H. Cunningham, and Y. Wilks. Named entity recognition from diverse text types. In Recent Advances in Natural Language Processing 2001 Conference, 2001.