A Metadata-based Architecture for Identification and Discovery of Services in SOA

Aluizio Haendchen Filho[1], Hercules Antonio do Prado[2] and Edilson Ferneda[2]
[1]*University College of Brusque - UNIFEBE, Brusque, Brazil*
[2]*Catholic University of Brasília, Brasília, Brazil*

Keywords: Resource-Oriented Model, SOA, Service Discovery, Metadata, Case-based Reasoning.

Abstract: An approach for resource identification, management, and service discovery in SOA is presented. The approach emphasizes an architectural model that allows representation, description, and identification of services, and is explored as a metadata repository. It is focused not only on Web Services, but also in all services existing in big companies' applications, including currently developed services and legacy system services, highlighting the importance of reusing fine granularity services. The model includes discovery procedures to find and retrieve candidates for services composition and reuse. These procedures adopt a Case-Based Reasoning approach, in which the services are considered as cases kept and indexed in a repository. Case matching is carried out by means of text mining techniques that allow finding the most appropriate service candidate with the desired requirements for a particular task.

1 INTRODUCTION

Service-Oriented Architecture (SOA) remains as the best option available for system integration and leverage of legacy systems (Lewis *et al.*, 2010). Technologies to implement SOA will certainly evolve to address emerging needs, but its basic concepts will remain.

The ability to compose applications and processes, as well as to assemble new functionalities from existing services is regarded as one of the most important benefits of SOA. Large and medium sized organizations can eventually have hundreds and even thousands of fine-grained procedures distributed across business applications.

Web Services and SOA are part of the solution for services composition and reuse, but applicability has been marked by difficulties in applying technical solutions. A large scale dissemination of SOA strongly depends on the coarse-grained services that are constructed and exposed to specific interfaces. Web services usually provide coarse-grained functionality such as customer lookup, as opposed to finer grained functionality such as customer address lookup (Lewis & Smith, 2007). Finer-grained operations result in large number of calls and increased network traffic, whereas coarse-grained operations may need to transmit unnecessary information (Fujita & Mejri, 2006). In order to design effective coarse-grained services, the fine-grained services and all other services need to be better described, represented and exposed to modelers, developers, and business technical analysts.

A metadata repository is vital to a company's ability to prosper in a service-oriented environment, but it must be built having specific business needs in mind to support business and technical users. Moreover, it must be built on a technologically sound architecture that will support future growth as applications evolve into business intelligence solutions. Additionally, mechanisms to facilitate the discovery of services are essential to enable the reuse and composition of new services (Marco, 2000).

A structural model and an infrastructure composed by a metadata repository for resource management and services discovering in SOA are presented, including techniques and tools for the identification and specification of services. The structural model is composed by a set of meta-classes that allow the representation and description of services and service components from different types and granularities for composition and reuse. The architecture and infrastructure model can be employed during various stages of the software development lifecycle. This allows designing and building of new tasks by a choreography of fine-grained services into

Filho, A., Prado, H. and Ferneda, E.
A Metadata-based Architecture for Identification and Discovery of Services in SOA.
In *Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS 2016) - Volume 2*, pages 298-305
ISBN: 978-989-758-187-8

composite business services. Discovery mechanisms use data mining techniques to allow finding the most appropriate service candidate with the desired requirements for a particular task or composition. The proposal was built over MIDAS (Haendchen Filho & Vasconcelos, 2015), a platform that uses SOA for development of distributed applications. This work extends the contribution of Haendchen Filho et al. (2015) by introducing the service discovery using *tf×idf* technique (Leskovec *et al.*, 2014).

2 SERVICES CLASSIFICATION

Classification helps to determine composition and layering, as well as to coordinate the construction of interdependent services based on hierarchy. The identification of business functions to be provided as services is a basic precondition for a detailed specification and implementation of services in SOA. For an effective SOA realization it is important that the service portfolio takes into account the correct service granularity. It helps not only in easing maintainability, but also in effective governance of the service portfolios (Saroh & Sahu, 2014).

Services are offered at different layers with a definitive granularity degree. Service granularity refers to service size and the scope of functionality that a service exposes. The service granularity can be quantified as a combination of the number of components/services composed by means of a given operation. The service should have the right granularity to accomplish a business work unit in a single interaction. Usually three categories of granularities are referred: services of coarse, medium, or fine granularity. The problem of determining the optimal service granularity is relevant since an increasingly number of applications has been built by assembling internal and external services (Manouvrier, 2008). Since most organizations have overlaping and redundancy both in business functionality and in data (e.g., Accounts Management) across different Lines of Business, it is essential to have a portfolio of business services which could be reused across multiple places.

A service would be regarded as coarse-grain if the size of exchanged messages grows and sometimes might carry more than needed data. On the other hand, if the service is fine-grained, then it can be excessively invoked, which introduces quality concerns and services outflow. A balance is hence required between level of abstraction, likelihood of change, complexity of the service, and the desired level of cohesion and coupling. High level business process functionality is externalized for large-grained services. Smaller-grained services help to realize the higher level of services. They are identified by examining the existing legacy functionality and deciding how to create adaptors and wrappers. (Arsanjani *et al.*, 2014).

Four different kinds of services are considered in the proposed platform: *(i)* services developed in the containers that use the internal service-oriented interfaces; *(ii)* legacy application services encapsulated in wrapper components; *(iii)* external web services of interest also encapsulated in wrapper components; and *(iv)* internal web services available for external applications or stakeholders such as customers and suppliers. The first three are described and shown in the internal structure of platform services, while the latter is represented in the internal UDDI registry.

3 THE RESOURCE-ORIENTED ARCHITECTURE

MIDAS architecture is composed by a Front-End Server (FES) and interconnected containers. Both FES and containers includes the Resource-Oriented Model (ROM) that is implemented by Catalog agent. A resource description is a metadata representation that makes possible for a human or software agent to discover service and provider entities (Bhuvaneswari & Sujatha, 2011). An important aspect of SOA is the extensive use of metadata (W3C, 2004). Resources are represented by meta-classes along with methods to manipulate and retrieve information about them. The proposed model provides meta-classes for defining different types of resources that may be found in an infrastructure environment.

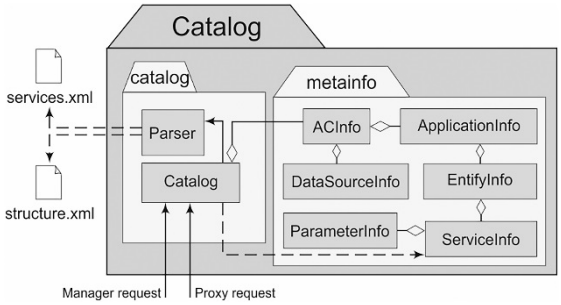

Figure 1: ROM and the metadata repository architecture.

Figure 1 shows the internal structure of Catalog agent placed in a container. It is composed by *catalog* and *metainfo* packages. In order to ease the application of the proposed infrastructure, the meta-

classes in the *metainfo* package are categorized in six major high-level classes: ACInfo, ApplicationInfo, EntityInfo, ServiceInfo, ParameterInfo, and DataSource info. These classes encapsulate information about elements of the structure and have a set of methods in order to obtain information about any element. The most important concerns that have been addressed in this model are the infrastructure management of resources, their internal organization, structure, classification and identification.

The resources representation and description are stored in two XML files: *structure.xml* and *services.xml*. In the *structure.xml* file, elements are arranged in a hierarchical parent/child entity set. In the *services.xml* file, the services are stored by name. These files enable two access modes: via the root structure or by direct access to the service. The first mode may be used in service discovery tasks. The second ensures efficiency, since service information can be quickly obtained using the service name as index, without traversing the structure.

Resources management and handling are performed by Catalog and Parser classes. The Catalog class receives requests which may be: *(i)* a message for resource structure updating; *(ii)* a resource structure loading is received when an AC is initialized; and *(iii)* a service location message is received to check if the service requested is available or not. The Parser class performs procedures for handling XML files, according to the Document Object Model (W3C, n.d.), in order to store and retrieve information.

3.1 Metadata Repository

SOA demands for a systematic identification of the information system functions to be implemented as services. Metadata publishing consists of making data element definitions and structures available to people and other systems. Metadata registries are frequently large and complex structures and require navigation, visualization, and searching tools.

The metadata is relevant because it fosters interoperability by requiring increased precision in the documentation of services. It also provides the semantic layer between the technical specification and the business analysts and developers. In simple terms, metadata translates the technical terminology used in IT systems in comprehensive terms for business analysts and modelers can understand (Marco, 2000).

The resource structure is described in the *metainfo* package, where each element is represented by a meta-class implemented as an entity. Each entity encapsulates information about the elements of the resource structure and provides a set of methods to retrieve information about the entity.

The ACInfo metaclass is the root of an aggregated class hierarchy representing the container resource structure. It keeps information about the container, such as its IP address, registration date, and data gathering statistics and metrics–for QoS report. ApplicationInfo metaclass encapsulates information about each application hosted in AC, such as agents, components, and services. The EntityInfo metaclass includes information about entities deployed in applications, such as business definition about the purpose of this entity, list of attributes, and operations, especially for reading and writing data meaningful to the company.

ServiceInfo metaclass provides detailed information of service, and its attributes compose the service descriptors that are used for services discovery. The services descriptors are: *(i) Name*, containing a representative name of the service, *(ii) Type,* referring to service classification, *(iii) Return,* which describes the data returning after the service execution, *(iv) Description,* that presents a summarized description on what the service does, *(v) Keywords,* containing significant words or expressions regarding the service functionality, *(vi) Implementation*, specifying how the service is available (ex. *Java service, Web service, wrapper component), and (vii) Classification*, described in the sub-topic Services Classification.

Information about services interfaces and the required parameters are provided by the ParameterInfo metaclass, which specifies the name, data type and additional parameters information. Input and output parameters of services must be specified and transferred into a formal interface definition. During this process a complete signature of operations (including e. g., data types, input and output parameters of service invocations) and the SOA standards expected to be used must be selected and defined.

DataSourceInfo metaclass provides information on the database tables. This description allows configuration of database drivers, as well as generic and specific services that can be used for manipulating data in database tables. One way to promote the reuse of legacy application functionality is to encapsulate their procedures in wrapper components. Databases, libraries, services, and other content sources can be wrapped in components (Sletten, 2009). In a shared information environment, it is highly relevant to allow access to metadata that describe a data source, regardless of the device and format in which it is stored.

3.2 Consolidate Resources Representation

FES Catalog agent performs the procedures for management and handling of the global resource structure. The responsibilities of Catalog agent in FES are different from the ones in the container. For example, since FES does not manipulate resources, there is no need to have the XML description of the resource structure. On the server, it only consolidates the representations of the resources allocated in containers, keeping them in cache memory and serialized files. The representation maintained in FES is a description of the consolidated hierarchy of resources that reflects the representation of all platform elements in a given moment.

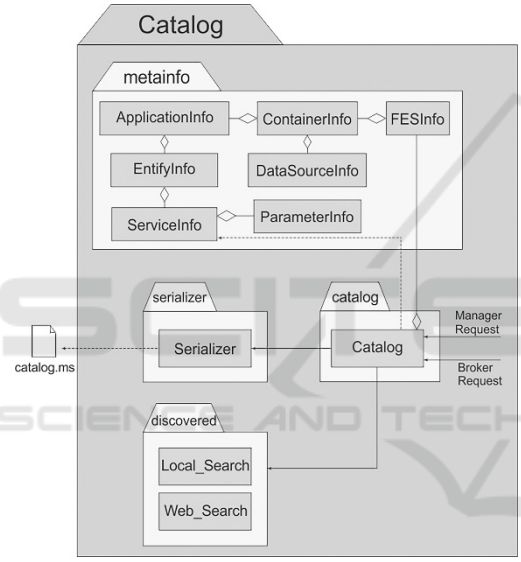

Figure 2: The FES Catalog agent.

Figure 2 shows its internal structure, which consists of three packages: *catalog*, *metainfo, discoverer* and *serializer*. All service requests are received by the Catalog agent through a single gateway. Requests for services visualization and discovery are forwarded from Manager to the Catalog. The Catalog class receives three message types: *(i)* a new container register or update is received when an AC wants to self-register on the platform or update its resource structure. In this case it sends all the elements of the resources structure encapsulated as parameters; *(ii)* for remote services location, requests are directed to the FES; Catalog agent is responsible for informing if the service is available or not; and *(iii)* services discovering requests.

The features of the consolidated structure are kept in cache memory and maintained in a serialized file. This procedure is carried out with the collaboration of a Serializer class, which is part of the Catalog internal structure. It performs all procedures related to the persistence of the resource structure. The data structures are consolidated and stored in serialized files and the representations are kept in cache memory. This increases efficiency, since quick service information can be obtained using the service name as an index, without traversing the structure.

The representation structure and content of *metainfo* package is completely similar to that described in the AC Catalog. The only difference is that FES meta-classes contain information about the overall structure of resources available in all containers. The *discoverer* package contains the classes responsible for services discovery. These procedures are performed by means of GUIs wizards available in the Manager agent. The Local_Search class performs procedures related to internal service discovery; it has a set of intra-classes which perform different discovery tasks. Internal service discovery is the process in which a user or application developer queries the central registry to learn about services location and specifications. The Web_Search class performs tasks related to the discovery of external web services in UDDI repositories.

4 SERVICES IDENTIFICATION

Services identification is a key component of most distributed systems and service oriented architectures. It is used to search services descriptions meeting certain functional or semantic criteria. There are two main user groups: business users and technical users. The first group typically has a business background and it gets needed information from metadata that enables them to identify and locate information about entities, services, databases tables, and attributes. Technical users may play many roles within an organization. They may be programmers, developers, system modelers, or senior analysts. Services identification procedures are primarily intended for developers' support, to facilitate the process of finding an appropriate service for a particular task.

The service identification process consists of a combination of top-down and bottom-up techniques of domain decomposition and existing asset analysis. In the top-down view, a blueprint of business use cases provides the specification for business services. This top-down process consists of decomposition of the business domain into its functional areas and subsystems, including the entities (Arsanjani *et*

al., 2014). In the bottom-up approach, the analyst departs from a service identifying the provider entity, where application and container it are located. The idea is to reach a context view from a service.

In order to reuse a service, clients need to know much more than a simple service name or the address of the service provider. Developers need to see a service as an interface, including methods that they will invoke in order to execute the service and their necessary parameters. The lookup service can be seen as a directory service, where services are found and viewed. There are two main reuse possibilities: *(i)* the service meets 100% of the requested specification or *(ii)* it is possible to reuse the ready implementation performing the necessary adjustments. Figure 3 shows a GUI wizard for service identification.

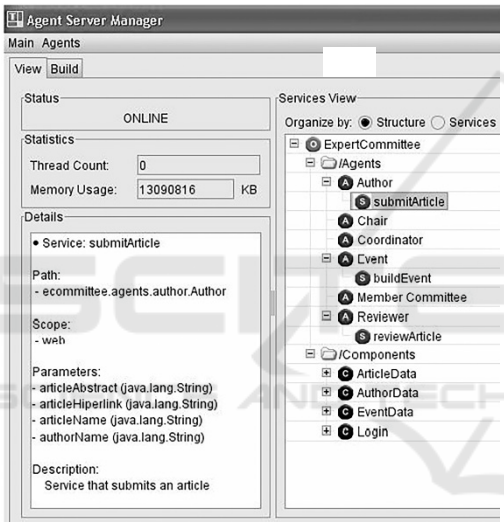

Figure 3: GUI wizard and the View perspective.

In the right side of the window, the Services View panel shows the resources organized by structure. The hierarchy is navigable and shows an agent-based application called Expert Committee for management of submissions and article reviews in a conference or workshop. Software agents have been introduced to assist the organizing committee in their responsibilities, which can be automated. The metadata contained in the repository can be viewed in the navigable hierarchy. In the left side of the window, the Details panel shows specifications of the selected service. The example presents the specification of the submitArticle service. The descriptors specifications include: *(i)* the service name (the entity type that provides the service); *(ii)* the path (URL) where the service is allocated; *(iii)* the scope informing if the service is local (in the container) or remote; *(iv)* name and type of the parameters; *(v)* a brief description of the service functionality; *(vi)* a return informing if any data type returns to the caller service; *(vii)* the keywords related to the service; and *(viii)* implementation, informing if the service is: implemented in Java, a Web service, or a legacy service encapsulated as a service in a component.

The service identification or discovery presents an important limitation when the amount of services increases. It is common to have hundreds or even thousands of services in an organization catalog.

5 SERVICES DISCOVERY

The increasing amount of web services currently available in the web has been targeted by a huge research effort aiming at reuse. Usually, they apply some Artificial Intelligent techniques like Case-Based Reasoning (CBR) and Ontologies. For example, Limthanmaphon and Zhang (2013) propose a solution to deal with coarse-grained services in which the composition is part of the service representation. Osman *et al.* (2006) present cases in a frame structure that is mapped to ontologies in order to enable a semantic retrieval of cases. Henni and Atmani (2012) apply CBR approach for service representation and retrieval. The grouping of services sharing similar functional requirements led ElBitar *et al.* (2014) to the notion of 'services community', in which the communities are identified and then a local search is performed to find a more adequate case. Each proposal has its merit in finding services by matching some requirement description in a web repository.

5.1 Proposed Approach

A CBR model is usually applied to solve problems by reusing solutions already defined for similar situations (Kolodner, 1993). For this, problems are represented as contextualized cases, keeping the information related to the circumstances in which the problem situation was considered partial or totally solved. A case is characterized by a set of descriptors for a problem situation associated with the respective solution.

In our approach, we argue that keeping local experience of services already used may largely enrich the information conveyed by the standard representation of service in WSDL format. For example, instead of searching in the open web repository we propose the creation of a local repository in which services already known by the developers can be

kept along with the information over its previous use. For this, the services must be represented by the standard information brought by WSDL representation plus a description related to the specific use of this service. This approach brings the need for evaluating similarity among the textual descriptions related to the narrative on the experience of the service use. Therefore, our proposal is a two-level recover. Firstly, CBR techniques are applied in order to recover the set of cases similar to the requirements description. Secondly, the cases are searched on the basis of their description, by applying a text mining procedure to find the nearest-neighbor case.

The process starts with a description of a service required by a developer. This description includes a functional account of the service representing the developer experience, beyond the usual descriptors like *name* and *parameters*. The system searches for a service in CB, guided by a given description, and then it retrieves a list of the best matching services. If a service satisfies the developer necessity, than it is applied. Otherwise, the alternatives are: *(i)* to search in the Web, *(ii)* to adapt a case from the retrieved case list, or *(iii)* to develop a new solution. In any case, the case base must be updated.

The CB structure includes the following descriptors: Name, Modifier, Parameters, Return, Service Description, and Keywords. However, for purpose of recovery cases, Modifier is not considered since it does not reflect a property that might be of interest in a specific search.

For similarity computation, we applied the aggregate degree of match from the CBR shell ReMind (Cognitive Systems, 1992). Considering that there are different types of descriptors (eg, free text for Description and a word for Return), the similarity function had to be adapted. Similarity for Name similarity is calculated by the Levenshtein distance. For Parameters, Return, and Keywords, Hamming distance was adopted. For Description similarity, two options were implemented: *(i)* a combination of co-occurrence of words and Hamming distance and *(ii) tf×idf* (Leskovec *et al.*, 2014) technique.

The Levenshtein distance between two features f_1 and f_2 is defined as the minimum number of insertions, deletions, and substitutions required to make $f_1=f_2$. The distance based on *tf×idf* for Description is calculated by means of a word matrix in which each service takes one row and each word from the Description argument takes one column. Each cell for a service/word receives the corresponding *tf×idf* calculated as the frequency for each word (here called term) in the argument of the service relative to its occurrences in the complete set of services. The Hamming distance between two features f_1 and f_2 is 0 for $f_1=f_2$ or 1, otherwise. Each distance is normalized for the interval [0,1].

5.2 Example of Application using Levenshtein Distance

Consider the case base shown in Table 1. Over this case base, a developer is looking for a service defined by the arguments *Arg* = {Address, String, String, "Recover address from Postal Office", "ZIP, Address"}. The Name, Return, Description, Keywords and Implementation descriptors are stored in the ServiceInfo meta-class, while the Parameter descriptor is stored in the ParameterInfo meta-class.

The results of similarity computation are shown in Table 2. The better adjusted case for the requirements specified in *Arg* is case 4. However, it is possible to have other cases that can be close to the requirements. So, the developer can ask to receive the cases until the k^{th} position in the matching ranking. In the example, for k=3, cases 4 and 7 would be submitted to the developer scrutiny in order to choose the most adequate case.

5.3 Example of Application using Tf×Idf

An example is the option *Arg* = {ControlledVocabulary, String, Boolean, "Need a controlled vocabulary for consistent communication for users natives in the English vocabulary", "Controlled Vocabulary, English Language"}. Table 3 shows that service 8 is the most similar to the specification given by the user. The distances from each vector to *Arg* are highlighted in black.

6 RELATED WORKS

This work is based on four main pillars: (*i*) SOA, (*ii*) a SOA-based middleware, (*iii*) the concept of ROM, and (*iv*) the intelligent services discovery encapsulated in the resource model. There are studies addressing each of the topics presented, none of them bringing together all of them into a single architecture.

WSA (W3C, 2004) includes the ROM in its reference architecture as part of a wider architecture. The resource model focuses on the features relevant to the concept of resource, disregarding the resource role in the context of web services. It takes the view that resources are a concept that underpins much of

Table 1: Case Base of Services.

#	Name	Parameter	Return	Description	Keywords
1	validate Password	String	Boolean	Checks a password for security issues	Password, Security, Method1, Method2
2	calculateCD	Integer	Integer	Applies an 10-Module algorithm to generate a check digit	Check digit, mod 10, Luhn Algorithm
3	calculateCD	Integer	Integer	Applies an 11-Module algorithm to generate a check digit	Check digit, mod 11, Verhoeff Algorithm
4	Retrieve Address	String	String	Retrieves the address for a given ZIP code	ZIP, Address retrieval
5	Currency Converter	[Real, String, String]	Real	Convert values between two currencies	Converter, Currencies
6	Metric Converter	[Real, String, String]	Real	Convert values between two metrics	Converter, Metrics
7	getAuthor Data	Integer	Hashmap	Retrieves data from a table using the ResultSet object of Java library	ResultSet
8	Activate Controlled Vocabularies	String	Boolean	Activate a controlled vocabulary according the language specified. A controlled vocabulary helps in keeping a consistent corporative communication	Controlled vocabulary, Corporative terms

Table 2: Distances/normalized distances from Arg features (using Levenshtein distance).

#	Name		Parameters		Return		Description		Keywords		Σ
	d	*Nd*	*d*	*Nd*	*d*	*Nd*	*d*	*Nd*	*d*	*Nd*	
1	24	0,63	1	0,33	1	1	8	0,47	6	0,86	3,29
2	18	0,47	1	0,33	1	1	10	0,59	7	1,00	3,40
3	18	0,47	1	0,33	1	1	10	0,59	7	1,00	3,40
4	**8**	**0,21**	**1**	**0,33**	**0**	**0**	**7**	**0,41**	**1**	**0,14**	**1,10**
5	24	0,63	3	1,00	1	1	9	0,53	4	0,57	3,73
6	22	0,58	3	1,00	1	1	9	0,53	4	0,57	3,68
7	**20**	**0,53**	**1**	**0,33**	**1**	**1**	**12**	**0,71**	**3**	**0,43**	**2,99**
8	38	1,00	0	0,00	1	1	17	1,00	6	0,86	3,86

Table 3: Distances/normalized distances from *Arg* features (using *tf×idf* distance).

#	Name		Parameters		Return		Description		Keywords		Σ
	d	*Nd*	*d*	*Nd*	*d*	*Nd*	*d*	*Nd*	*d*	*Nd*	
1	36	0,97	0	0	0	0	4,14	0,82	6	1	2,79
2	31	0,84	1	1	1	1	4,79	0,94	5	0,83	4,62
3	31	0,84	1	1	1	1	5,07	1,00	5	0,83	4,67
4	34	0,92	0	0	1	1	3,95	0,78	5	0,83	3,53
5	37	1,00	1	1	1	1	3,84	0,76	4	0,67	4,42
6	35	0,95	1	1	1	1	3,84	0,76	4	0,67	4,37
7	33	0,89	1	1	1	1	4,39	0,87	3	0,50	4,26
8	**8**	**0,22**	**0**	**0**	**0**	**0**	**3,76**	**0,74**	**2**	**0,33**	**1,29**

the web and much of web services. A discovery service enables agents to retrieve web service-related resource descriptions, and is used to publish and search for descriptions meeting certain functional or semantic criteria.

The main difference between our and WSA models is that the latter considers Web service as the most important resource for its purposes and UDDI registry as the target for the discovery service. For composition and reuse, we consider other finer-grained services as well as atomic services using service-oriented interfaces. Our main motivation is the difficulty of WSDL to describe complex business services typically formed by fine-grained services composition. The structural model enables all the resource types to be defined and exposed in the architecture for composition and reusing.

Our resource model is part of a wider architecture and has been validated and used to demonstrate case studies. In these prototypes the applicability of

the presented mode has been confirmed. Regarding the intelligent service discovery, Limthanmaphon and Zhang (2013) proposed a service discovery process that adopts fine-grained services as cases, including the relations of dependency between them. Osman *et al.* (2006) also present a CBR proposal for services discovery using ontology to structure the cases representation. ElBitar *et al.* (2014) discuss the semantic-based approaches, focusing on CBR models. They propose the concept of "community service" to represent the group of Web services functionally similar. These approaches use Web as an information source. They try to overcome the limitation of the syntactic discovery based on UDDIs. No concern with respect to the local experience in applying the services is emphasized.

7 CONCLUSIONS

It is presented two contributions: *(i)* an architectural model for a metadata repository for SOA, that enables the discovery of different types of services for composition and reuse; *(ii)* a CBR system in which case matching is carried out by means of text mining techniques, allowing one to find the most appropriate service candidate with the desired requirements for a particular task or composition. There are only a few papers exploring architectural models for metadata repository in the context of services identification and discovery in SOA. CBR has been already used in some approaches to services discovery, but we have no knowledge of adoption of text mining techniques as a support for case matching, such as is presented in this paper.

REFERENCES

Arsanjani, A., Zhang, L.-J., Ellis, M., Allam, A., Channabasavaiah, K., 2014. Design an SOA solution using a reference architecture. Improve your development process using the SOA solution stack. *IBM DeveloperWorks*.

Bhuvaneswari, N. S., Sujatha, S., 2011. *Integrating SOA and Web Services*. River Publishers.

Cognitive Systems, 1992. *ReMind Developer´s Reference Manual*, Cognitive Systems.

ElBitar, I., Belouadha, F.-Z., Roudies, O., 2014. A CBR based approach for web service automatic discovery. *Journal of Theoretical and Applied Information Technology*, 62(1):237-247.

Fujita, H., Mejri, M. (Eds.), 2006. New trends in software methodologies, Tools and Techniques. *Proceedings of the Fifth SoMeT_06*. Frontiers in Artificial Intelligence and Applications, 147.

Haendchen Filho, A., Vasconcelos, C., 2015. Development Of Intelligent Enterprise Applications Using Multi-Agent Systems. *Australian Journal of Basic and Applied Sciences*, 9(11):444-450.

Haendchen Filho, A., Prado, H. A., Ferneda, E., 2015. A Resource-Oriented Model for Services Management and Discovering in SOA. *IEEE International Conference on Services Computing (SCC)*.

Henni, F., Atmani, B., 2012. Dynamic Web Service Composition: Use of Case Based Reasoning and AI Planning. *4th International Conference on Web and Information*, p. 22-29.

Kolodner, J. L., 1993. *Case-Based Reasoning*. San Francisco, CA: Morgan Kaufmann.

Leskovec, J., Rajaraman, A., Ullman, J., 2014. *Mining of massive datasets*. 2nd ed, Cambridge University Press.

Lewis, G. A., Smith, D. B., 2007. Four pillars of Service-Oriented Architecture. CrossTalk: *The Journal of Defense Software Engineering*, p. 10-13.

Lewis, G. A., Smith, D. B., Kontogiannis, K., 2010. A Research Agenda for Service-Oriented Architecture (SOA): Maintenance and Evolution of Service-Oriented Systems. Technical Note, CMU/SEI-2010-TN-003. Carnegie Mellon University.

Limthanmaphon, B., Zhang, Y., 2013. Web Service composition with Case-Based Reasoning. *14th Australian Database Conference*, 17:201-208.

Manouvrier, B., 2008. *Integration applicative EAI, B2B, BPM et SOA*. ISTE Ltd / John Wiley & Sons, Inc.

Marco, D., 2000. *Building and Managing the Meta Data Repository: A Full Lifecycle Guide*. John Wiley and Soons Inc.

Osman, T., Thakker, D., Al-Dabass, D., 2006. Semantic-driven matchmaking of Web Services using Case-Based Reasoning. *IEEE Int. Conf. on Web Services*.

Saroh, M., Sahu, S., 2014. Review on "Service Granularity In Service Oriented Architecture". *International Journal of Research in Engineering and Technology*. 3(7):449-455.

Sletten, B., 2009. Resource-Oriented Architecture: The Rest of REST. *InfoQ*. Retrieved January 2015 from http://www.infoq.com/articles/roa-rest-of-rest.

World Wide Web Consortium [W3C], 2004. *Web Services Architecture*, W3C Working Group Note 11.

World Wide Web Consortium [W3C], n.d. *W3C Document Object Model (DOM). XML DOM Tutorial*.