Ontology Matching in the Context of Web Services Composition

Tyl, Pavel
2010

Ontology Matching in the Context of Web Services Composition

Post-Graduate Student:
ING. PAVEL TYL

Institute of Computer Science of the ASCR, v. v. i.
Pod Vodárenskou věží 2
182 07 Prague 8, CZ

Faculty of Mechatronics, Informatics and Interdisciplinary Studies
Technical University of Liberec
Hálkova 6
461 17 Liberec 1, CZ

pavel.tyl@tul.cz

Supervisor:
ING. JÚLIUS ŠTULLER, CSC.

Institute of Computer Science of the ASCR, v.v.i.
Pod Vodárenskou věží 2
182 07 Prague 8, CZ

stuller@cs.cas.cz

Field of Study:
Technical Cybernetics

This project is partly realized under the state subsidy of the Czech Republic within the research and development project “Advanced Remediation Technologies and Processes Center” 1M0554 – Programme of Research Centers PP2-DP01 supported by the Ministry of Education, under the financial support of the ESF and the state budget of the Czech Republic within the research project CZ.1.07/2.2.00/07.0008 – ESF OP EC “Intelligent Multimedia E-Learning Portal” and cofinanced from the student grant SGS 2010/7821 “Interactive Mechatronics Systems Using the Cybernetics Principles”.

Abstract

Web services became one of the best means for web application interoperability. There is a need to have a scalable and extensible model to deliver distributed information and functionality integrated as independently provided, interoperable services in a distributed environment. Several distributed services can be dynamically composed (chained) as a new service to accomplish specific tasks. Such a model of service composition (chaining) is one of the most important research topics of next generation web services.

This paper discusses possibilities of using ontology matching techniques for web services interoperability and composition, describes such processes, explain their difficulties and propose a model for web service composition based on suitable ontology matching techniques.

1. Motivation

Let’s suppose this motivation scenario:
We want to deliver some electronic product from a web shop to some address by a shipping service. Online electronic shop service provides its output description in some ontology. Shipping service uses a second ontology for its input description. Then the matching of these ontologies could be used for:

- checking that what is delivered by the first service, e. g., a `DVD_Player`, matches what is expected by the second one, e. g., some `Object` (shipping service does not accept life animals),
- verifying preconditions of the second service, e. g., `Size` in centimeters, etc.

We can see only two parts of a chain in this short example, but there could be many more. For example there are web services able to compare products (e. g., `DVD_Players`) from different data sources (catalogues), some web services do it even more sophisticated using user preferences, etc.

2. Introduction

2.1. Ontology matching

Ontology matching is the process of finding “correspondences” (also called relationships [3]) between elements within different ontologies which have to be (semantically) compared and, eventually, joined. The output of the matching process is a set of such correspondences between two (or, in general, more ontologies) called an *ontology alignment*. The “oriented” version of an ontology alignment is an *ontology mapping*.

Given two source ontologies o and o', an input (“preliminary”) alignment A, a set of *parameters* (e. g., threshold) and *resources* (e. g., provenance metadata), the **matching process** (see Fig. 1) can be described by function f returning a *new alignment* A' between ontologies o and o':

$$A' = f(o, o', A, p, r).$$

Ontology matching is in most cases performed *manually* or *semiautomatically*, often with support of some *graphical user interface*. Manual specification of ontology parts for matching is *time consuming* and moreover *error prone process*. It results in a strong need for development of faster and/or less laborious methods, which can process ontologies at least semiautomatically.

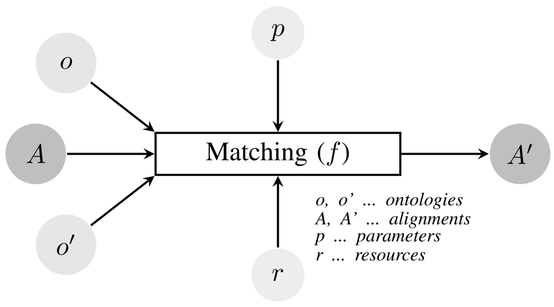


Figure 1: Schema of a matching process [3].

2.2. Web services

A web service is a network accessible interface to web application functionality. It is described in machine-readable format, most often in standardized web service description language, WSDL [15]. Way of communication between other computers and web service is specified in the web service's description with the help of Simple Object Access Protocol, SOAP [12]. SOAP messages are transfered by well-established protocols[1]. SOAP and WSDL protocols have easy machine-readable XML [17] syntax. Both SOAP and WSDL were designed to be independent on selected version of XML language, but obligated to be XML compatible.

The W3C [13] defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards" [14].

Web services expose their interfaces to the web so that users (agents) can invoke them. **Semantic web services** provide a richer and more precise way to describe services through the use of knowledge representation languages and ontologies [4], e. g., OWL-S [11] or WSDL-S [16].

2.3. Web service composition

Web service discovery and integration is the process of finding web service able to deliver a particular service and composing several services in order to achieve a particular goal [8].

Web services are often designed to be independent and replaceable and, therefore, web service processors are able to incorporate new services in their workflows and customers can dynamically choose new and more promising services. For that purpose, they must be able to compare the descriptions of these services (in order to know if they are really relevant) and to route the knowledge they process (in order to compose different services) by routing the output of some service to the input of another service.

Both for finding the appropriate service and for interfacing services, some data "mediator" is important as a bridge between different vocabularies [9]. Based on the correspondences between the terms of the descriptions, mediators must be able to translate the output of one service into a suitable input for another service (see Fig. 2).

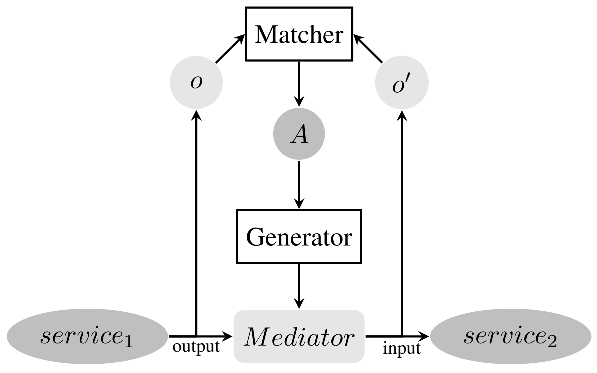


Figure 2: Web service composition. In this case it is useful to (1) match relevant parts of ontologies o and o', thus resulting in alignment A and (2) $generate$ a $mediator$ between $service_1$ and $service_2$ in order to enable transformation of actual data [3].

The headstone of a mediator definition is an alignment between two ontologies. And this can be provided through matching the corresponding ontologies either offline when someone is designing a preliminary service

[1]Reason for SOAP messages "encapsulation" is an absence of trust from existing systems.

composition, or online (dynamically) [5], when new services are searched for completing a request.

3. Web service composition by ontology matching

There are two possibilities one could use ontology matching techniques for web service composition:

1. Entire web service is described by the service ontology (WSMO[2]).
2. Web service is described by the traditional means and only its inputs and outputs are described by the ontologies.

Ontologies classifying and describing services are called service ontologies. According to our opinion it is not necessary to describe web services by ontologies (i. e., using WSML[3] [5]), because inner behavior of a web service *need not* (sometimes rather *must not*) be always transparent or accessible. But at least web services' outputs and inputs have to be described using ontologies for successful application of ontology matching techniques.

Every web service has its input(s) and output(s), in our case described as input ontologies $o(in\_s_1)$ and $o(in\_s_2)$ and output ontologies $o(out\_s_1)$ and $o(out\_s_2)$ (see Fig. 3). They can be part of the *web service I/O interface* or can be stored outside the web service itself.

Web service in our model is devided into two main parts – its *internal structure* and a *repository*. The *internal structure* is responsible for functional achievment of the exposed service, finding direct or intermediate answers. If the web service is able to provide a direct answer (reply to the *primary request*), the input ontology $o(in\_s_1)$ is processed in the *internal structure* and results are transferred to the output ontology $o(out\_s_1)$. In the case of intermediate answer, if the *web service 1* is compliant to be a part of a chain, $o(out\_s_1)$ is produced and devolved upon a *web service chain repository* with a goal of searching for the best available web service for the chain, so that its appropriate alignment (e. g., A_{12}) is in the *alignments repository*.

When a preliminary alignment A_{12} exists (provided manually or by (semi)automatic means), it should be stored in the *alignments repository* for simple identification of reuse opportunities (see Sec. 3.1). Input/output ontologies $o(out\_s_1)$ and $o(in\_s_2)$, alignments $A_{12}, A_{13}, \ldots, A_{1n}$ and saved readymade *web service chains* are in the web service repository identified by their URIs[4]. It allows interaction with other services in order to negotiate operations the current service just cannot provide (e. g., when the current service is not available). Therefore at least the alignments (or the whole repository) should be always exposed in the same way as the inputs and the outputs of the web service.

If the suitable service (e. g., *web service 2*) is found and preliminary alignment exists, $o(out\_s_1)$ and $o(in\_s_2)$ are checked for their compatibility in a *compatibility checker* and if they pass, $o(out\_s_1)$ is easily converted in an *I/O converter* by using stored alignment A_{12} into $o(in\_s_2)$ and the request is passed on. Successful conversion and checks should be stored and cached.

If there is no alignment related to *web service 2* in our *alignments repository*, traditional matching methods (*Matcher* in Fig. 3) or manual matching have to be used. If related alignment exists, we can successfully apply an alignment reuse methods (see Sec. 3.1). According to [7] there are four matchmaking functions based on which web services can be chained: *Exact*, *PlugIn*, *Subsume* and *Intersection*. Otherwise (*Disjoint*), services are incompatible:

- **Exact** – if the output parameter out_s_y of s_y and the input parameter in_s_x of s_x are equivalent concepts (e. g., `DVD_Player` from our motivation example could be certainly delivered, because it is an `Object` and its `Size` is less than maximal allowed),
- **PlugIn** – if out_s_y is a subconcept of in_s_x (e. g., `DVD_Player` will be be delivered, if a shipping service is able to deliver whatever we want, any `owl:Thing`),
- **Subsume** – if out_s_y is a superconcept of in_s_x,
- **Intersection** – if the intersection of out_s_y and in_s_x is satisfiable,
- **Disjoint** – if the out_s_y and in_s_x are incompatible.

With *Exact* and *PlugIn* functions we are always able to match required web services, the matcher can fail in case of *Subsume* and *Intersection*.

[2]Web Service Modelling Ontology – http://www.wsmo.org.
[3]Web Service Modelling Language.
[4]Uniform Resource Identifier.

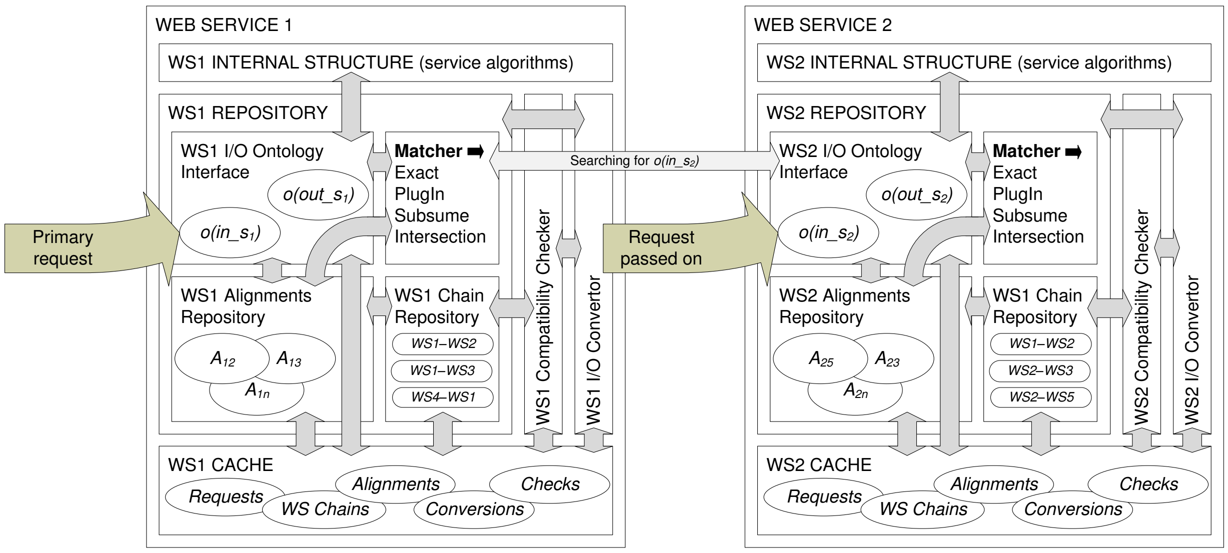


Figure 3: Proposed model for the web service composition with using of an ontology matching.

3.1. Candidate matching techniques for web service composition

It would be nice if we could always automatically create input and output ontology alignments at runtime. But it is not an easy task in the case of heterogenous web service compositions. In addition such algorithms should be fast enough, there is no time for tuning parameters, manual corrections, etc. Therefore in our model we suppose at least preliminary ontology alignments of $o(out\_s_y)$ and $o(in\_s_x)$ at design time. Consequently we can always take an advantage of them. And at this moment an alignment reuse can come on scene.

Alignment reuse is motivated by an idea that many ontologies that should be matched are similar to already matched ones, especially if they describe the same domain(s). Ontologies from the same application domain usually contain many similar elements, typical for this domain. Therefore their mappings can provide good reusable candidates.

At first, matching problems are decomposed, then a set of ontology fragments is generated and finally previous match results can be applied at the level of ontology fragments rather than at the level of the whole ontologies [3]. According to [2] alignments of ontologies (e. g., o_y and o_x) can be saved in a repository in three possible forms:

- **Direct mappings** ($o_y \leftrightarrow o_x$) – ideal for reuse, one or multiple mappings are already available for the given match problem. Such mappings represent the shortest possible mapping paths, which do not involve any intermediary ontologies.
- **Complete mapping paths** ($o_y \leftrightarrow o_i$, $o_i \leftrightarrow o_x$ or $o_j \leftrightarrow o_y$, $o_x \leftrightarrow o_j$) – such mapping paths consist only of existing mappings.
- **Incomplete mapping paths** (same as complete, but i. e., $o_i \leftrightarrow o_x$ and $o_j \leftrightarrow o_y$ are to be matched) – the default match operation is first applied and missing alignments can be computed with less effort than directly matching the input ontologies.

All match results are compared (e. g., average similarity in the path, expected computational effort expressed by the path length, etc.) and ontologies composed.

Although alignment reuse seems to be the most important technique in the proposed model (more than technique we could call it matching strategy), there are some other basic techniques that cannot be omitted in the web service composition. In the following we list some of them together with the reason for their use:

- **Internal structure (constraint) based techniques** – before creating an ontology alignment, but much more for later use, we can do a verification of criteria as the set of the entity properties (e. g., their multiplicity), the range and the domain of the properties, cardinality, datatypes, etc. These techniques are easy to implement and if the ontologies ($o(out\_s_y)$ versus $o(in\_s_x)$) pass them, it will provide a basis on which other parts of an application can rely.

- **External ontologies based techniques** – external reference ontology (o_{ext}) can provide a common ground on which an alignment can be based. It can help in the case of disambiguity of multiple possible meanings of terms in given domain of interest. For example an alignment between $o(out\_s_y)$ and $o(in\_s_x)$ can be derived from two other alignments with external ontology ($A(o(out\_s_y), o_{ext}$ and $A(o(in\_s_x), o_{ext})$[5]. External ontology is in the most cases a general reputable upper-level ontology (e. g., FMA[6] in medicine).
- Further we can use **relational structure techniques** (e. g., taxonomy relations), **propositional** and **description logic techniques** (these techniques cannot find an alignment alone, but when alignment is generated, we can ensure its completeness and consistency), etc.

4. Potential problems of web services composition

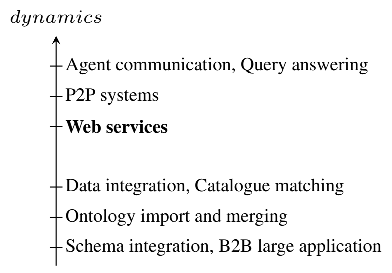
Issues worth mentioning to deal with when composing web services are:

Third-party sources – Two things remain unchanged with the web services in general:

- they use third-party sources and
- they have questionable reliability.

This is not to say that web services are unreliable, but it simply means that we have not a primary control of the source for our web application. When our sources are offline, our web service or web application is also offline. One way to avoid this problem is to keep an actual cache of all queries issued to our data sources in case of a service failure.
Caching is a good idea in general because it will definitely speedup repeated requests.

Rate limiting – Many public service interfaces may have to limit the number of requests an application or user can make within certain period of time. (This can be done by tracking the number of requests made by a single IP address or the system may require authentication.) This is another issue that could be partially or fully solved by request caching. Fig. 4 shows the position of web services among other web applications. As can be seen, they are supposed to be dynamic, but in contrast to P2P systems they should stay always correct.



dynamics
Agent communication, Query answering
P2P systems
Web services
Data integration, Catalogue matching
Ontology import and merging
Schema integration, B2B large application

Figure 4: Example applications ordered by their *dynamics*. Space under semantic web services shows that three top applications are considered to be dynamic.

Reliability – Keeping the current cache of recent requests can help keeping our service online until our sources are back online. If more than one public service interface is available to provide the information, our composite web service requires then a fallback mechanism to be implemented. It allows our web service to switch to another source until our primary source has been reestablished or to find another reliable (data) source forever.

Vendor locking – This could be a huge problem in the future as more and more web service compositions will be created. What to do if a public application interface that serves alignments to thousands of web services and web applications suddenly goes offline for one day or forever or starts charging for their service? It is therefore necessary to share accessible sources or prepare mechanisms for rapid finding of other appropriate services.

Licensing restrictions – Some public web services restrict for what we can use them and sometimes which web services can we use together with them. We have to thoroughly read restrictions which may apply before adopting them to our web service chain [7]. This problem is similar to the above one and has the same possible solutions.

5. Conclusion and future work

In this paper the possibility of using ontology matching techniques in web service composition is presented and a complex model for such composition is proposed. There are many different applications which require or could take an advantage of ontology matching. But in

[5] Here we can omit the input alignment, the resource and the parameters.
[6] Foundational Model of Anatomy – http://sig.biostr.washington.edu/projects/fm.
[7] One more disadvantage is that these restrictions or rules can be time invariant and practically stochastic.

the comparison with traditional applications such as information or schema integration, web service composition has its specific requirements – after preliminary steps (creating and processing alignments) it should be automatic and dynamic enough. Therefore we have to store these alignments and find the way how to replace them if necessary.

The next step I would like to work on is a design and implementation of an application according to the proposed model that will be able to compose e-learning systems for advanced testing (could be seen as web services) with the help of ontology matching (or, in general, ontology integration) techniques.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web Services. Concepts, Architectures and Applications". Springer, Berlin, 2004. ISBN 3-540-44008-9.

[2] H. Do, "Schema matching and mapping-based data integration". PhD thesis, University of Leipzig, Leipzig, DE, 2005.

[3] J. Euzenat and P. Shvaiko, "Ontology Matching". Springer-Verlag, Berlin/Heidelberg, 2007. ISBN 978-3-540-49611-3.

[4] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domungue, "Enabling semantic web services: the web service modelling ontology". Springer, Heidelberg, 2004. ISBN 978-3-540-34519-0.

[5] F. Guinchiglia, F. McNeil, and M. Yatskevich, "Web service composition via semantic matching of interaction specifications". Technical Report DIT-06-080, University of Trento, 2006.

[6] D. McCandless, L. Obrst, and S. Hawthorne, "Dynamic Web Service Assembly Using OWL and a Theorem Prover". In Proc. 3^{rd} IEEE International Conference on Semantic Computing, Berkeley, USA, 2009.

[7] F. Lécué, A. Delteil, and A. Léger, "Applying Abduction in Semantic Web Service Composition". In Proc. 2007 IEEE International Conference on Web Services, p. 94–101, IEEE CS, 2007.

[8] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities". In Proc. 1^{st} International Semantic Web Conference (ISWC), volume 2342 of LNCS. p. 333–347, Chia Laguna, IT, 2002.

[9] D. Roman, H. Lausen, and U. Keller, "Web service modeling ontology standard (WSMO standard)". Working Draft D2v0.2, WSMO, 2004.

[10] OWL – Web Ontology Language / W3C Semantic Web Activity [online]:
http://www.w3.org/2004/OWL.

[11] OWL-S – Semantic Markup for Web Services [online]:
http://www.w3.org/Submission/OWL-S.

[12] SOAP – Simple Object Access Protocol [online]:
http://www.w3.org/TR/soap.

[13] W3C – World Wide Web Consortium [online]:
http://www.w3.org

[14] W3C – Web Services Glossary [online]:
http://www.w3.org/TR/ws-gloss

[15] WSDL – Web Services Description Language [online]: http://www.w3.org/TR/wsdl.

[16] WSDL-S – Web Service Semantics [online]:
http://www.w3.org/Submission/WSDL-S.

[17] XML – Extensible Markup Language / W3C XML Activity [online]:
http://www.w3.org/XML.