Model Matching for Web Services on Context Dependencies

Georgia M. Kapitsaki
University of Cyprus
P.O. Box 20537
1678, Nicosia,Cyprus
gkapi@cs.ucy.ac.cy

Achilleas P. Achilleos
University of Cyprus
P.O. Box 20537
1678, Nicosia,Cyprus
achilleas@cs.ucy.ac.cy

ABSTRACT

Model matching has been applied to different fields of Model Driven Engineering research and usually concerns models depicting the same information in order to detect their evolution process or elements on which model migration is needed. In the current paper a different approach is followed by comparing models that depend on each other on a more specific basis: matching for Web Service models using dependencies on context information is addressed. The models follow the notation of an appropriate Web Service metamodel that captures Web Service properties. The models are represented as typed attributed graphs and their comparison is performed on an introduced variant of the Similarity Flooding algorithm, which is usually applied on schema matching. The application of the matching process is demonstrated through a number of constructed and existing Web Service description graphs extracted from online service registries.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.13 [**Software Engineering**]: Reusable Software—*Reuse models*

General Terms

Design

Keywords

model matching, model comparison, web services, context, context-awareness

1. INTRODUCTION

Model transformation and model matching constitute two important fields in the Model Driven Engineering (MDE) research that has emerged from OMG's Model Driven Architecture (MDA) [10], but with a much wider sense. In many cases model transformation is related to model matching used in diverse domains in order to discover correspondences between model elements. In the existing literature the common use case is to apply model matching on models depicting the same kind of information, expressed usually in different versions of the same model. These models may be based on the same metamodel notation or conform to different metamodels. In the latter case, model matching on the metamodel level also needs to be addressed [23]. When versions of the same model are considered model matching is performed for purposes of model or system evolution, which is a useful activity in the software engineering industry for expressing the evolution of software systems. Model matching can also be used for identifying elements on which model merging can be performed. This case is used for purposes of model migration to new systems.

Model matching or model comparison has been addressed from the above perspective in many previous works by providing generic or specialized solutions (e.g., SiDiff [21]). In this paper we address the issue of model matching for the case of Web Services (WSs) and more specifically for context-aware Web Services capturing context adaptation, i.e., adapting Web Services to context information. Context-awareness refers to applications and services with the capability of adapting themselves proactively to the information on user, environment and application context. Context can be used to characterize any information relevant to an entity and its surroundings [1]. Examples can be found in the user location, current activity, health condition and weather conditions [12]. This meaning of context is assumed in the current work.

Various context-aware Web Service systems exists in the literature [22], whereas context-aware Web Services are usually given in modeling notation. Although context-awareness in Web Service systems has spread, handling development at the modeling level, especially when existing services are employed in order to build new, larger applications, is missing. Existing Web Service models can be combined in order to result in context-aware applications consisting of reusable models. The existing work tackles this gap by presenting a solution in model matching for Web Services for context-awareness purposes. In order to make the need for this matching on Web Services more explicit the categories of *Business Web Services (BWSs)* and *Context Web Services (CWSs)* are considered as introduced in previous works [6, 7]. In short, BWSs require context information for their functionality (e.g., tourist services retrieved based on the user location), whereas CWSs offer context information (e.g., providing information on weather conditions based on sensor data). Note that there are cases, where a WS can have both roles, i.e., BWS and CWS, depending on the scenario of use. Both categories can be encountered in software systems developed either internally or as reusable services from individual providers indexed in service registries, such as seekda[1] and webservicex[2].

In this paper an adapted version of the Similarity Flooding algorithm [14] for handling the problem of Web Service model matching for context adaptation purposes is being proposed. The matching is performed with the following steps: 1) Input model representation as typed attributed graphs, 2) Construction of the similarity propagation graphs for context-related elements of the models, and 3) Application of the variant of the similarity flooding algorithm. The main part of the matching procedure is found in step 3, where techniques related with label matching and similarity detection employing different matching algorithms are used. The similarity functions considered are widely employed in the field of bioinformatics (e.g., Levenstein distance or the Smith Waterman algorithm for string matching).

The rest of the paper is structured as follows: Section 2 presents the adapted similarity flooding algorithm by analyzing the different steps of the procedure. Section 3 is dedicated to the employed similarity functions ranging from string matching to semantic concepts. A demonstration of the application of the algorithm on Web Service models is discussed in Section 4. The related work of the field is presented in Section 5. Finally, Section 6 concludes the paper.

2. ADAPTED SIMILARITY FLOODING

A variant of the similarity flooding algorithm [14] is proposed for performing the model matching between Business Web Service and Context Web Service models. Both input models to be matched are represented using the WS metamodel present in the modified version of the ContextUML metamodel [16] introduced in [8]. The ContextUML metamodel is divided into two parts: one part is dedicated to the description of Web Service-relevant information that can be exported from the WS specification description in Web Service Definition Language (WSDL) and the other is used for describing context information. As aforemementioned in the current work only the first part is employed containing elements present in WSDL documents as presented also for WS composition purposes in a Unified Modeling Language (UML) metamodel in [17].

2.1 Model Representation

The aforementioned Web Service metamodel has been transformed to the typed attributed graph of Fig. 1. Typed attributed graphs $ATG = (G; D)$ are used extensively in graph transformations [4]. In theory they consist of an E-graph G together with a DSIG-algebra D (data signature algebra). In practice they constitute an abstract syntax graph of a model. In the current work the WS metamodel has been used as input for the construction of the corresponding typed attribyted graph. The graph consists of:

- square graph nodes (e.g., *BusinessService*, *WSOperation*)
- data nodes (only the *String*data node is present in the metamodel)

[1]http://webservices.seekda.com/
[2]www.webservicex.net/

Nodes are connected with edges depicting either: the relations between graph nodes with a similar functionality as the relationships in UML modelling, or the type of attributes linked with a graph node (e.g., graph node *Message* has an attribute *name* of type *String*). The reader can refer to previous work of the authors for more information on the metamodel [7].

In order to introduce the elements that participate in the metamodel, two sample Web Service models are depicted in Figs. 2 and 3. The first model refers to a BWS, namely *TemperatureWS*, that provides temperature information for specific locations through its *getTemperature* operation. The input of this operation is the current city, where the requester is located, as expressed in the input message *Part* element named *cityName*. This input information refers to contextual information and expresses the dependency of the BWS to context. This type of dependency is called *parameter injection* [9]. In parameter injection one or more input parameters of the input message of the Web Service correspond to context data. For the proper WS execution this context data needs to be available.

The second WS model is an example of the CWS *LocationWS* that provides information on the current location of the requester. Specifically, through the two offered operations, i.e., *getCountryByIP* and *getCityByIP*, the current country and the current city can be retrieved respectively. Both operations require the IP address of the requester's machine as input and return the information in a *String* representation as in the *ValueObject* type of the output message *Part*.

From the model representation it can be deducted that the *LocationWS* CWS may be used to retrieve the city name context information needed by the *TemperatureWS* BWS for the proper invocation of its getTemperature operation as required by parameter injection. Apart from this adaptation type, a second adaptation addressed is found in *operation selection* also introduced in [9]. Operation selection corresponds to the case, where the selection of the BWS operation to be invoked depends on context information. Consider for instance the *OurProductsWS* (Fig. 4), which exposes a number of operations for retrieving prices for the products of the company *OurProducts*. Since the produtcs are distributed in various countries, the applicable prices in each country are captured in separate operations: *getGermanyPrice* and *getItalyPrice* retrieve prices in Germany and Italy respectively. Depending on the location of the requester a choice is made on the operation to be called. This is an operation selection case for the two operations depending on context. Generally, operation selection is used when a WS exposes more than one operations that provide the same functionality in different ways or with different parameters (in number and meaning). The decision, on which operation to invoke, depends on the current context data.

2.2 Compatibility Graph

In order to express the compatibility between elements of the two models, the Pairwise Connectivity Graph (PCG) introduced in the similarity flooding algorithm [14] has been employed. The PCG contains nodes from models B and C to be matched, where $((x, y), p, (x', y')) \in PCG(B, C)$, when $(x, p, x') \in B$ and $(y, p, y') \in C$ with x, x', y, y' cor-

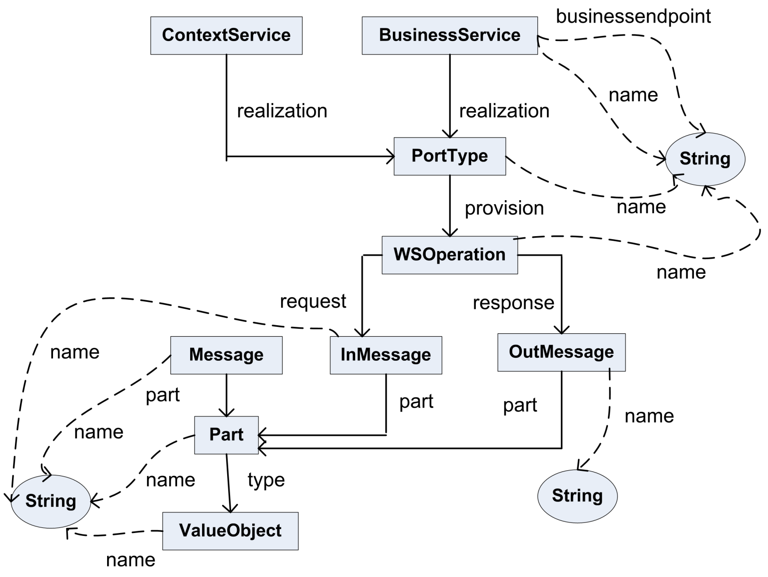

Figure 1: Typed attributed graph for the modified *ContextUML* metamodel.

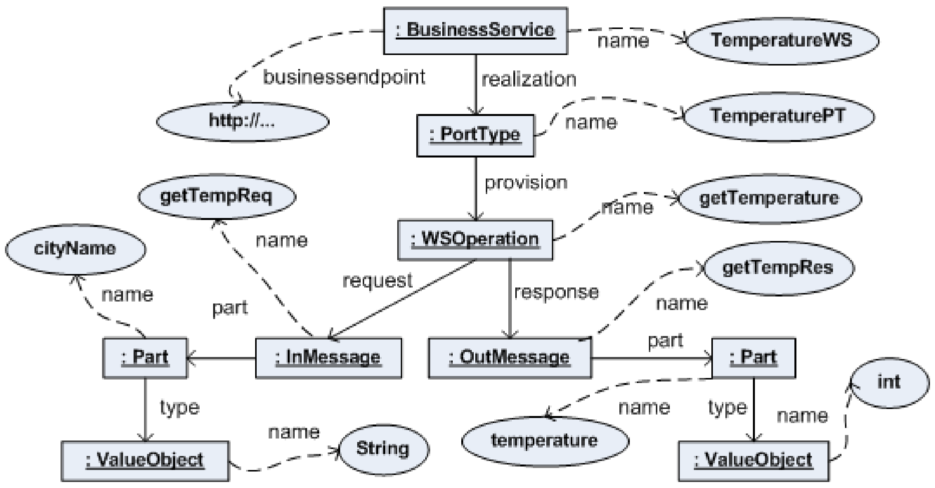

Figure 2: Typed attributed graph of *TemperatureWS* Business Web Service.

responding to nodes and p to an edge in the input model graphs.

In the original similarity flooding algorithm a unique PCG is sufficient to express the compatibility between two models. However, in the current problem addressed the matching is not only applied on similar graph elements on an exhaustive basis for all graph nodes. When targeting the context adaptation cases, specific matching rules need to be defined and expressed as part of the PCG. Therefore, the PCG concept is extended by multiple PCGs with each one expressing a different matching rule. Each PCG contains only the necessary elements participating in the rule; the rest of the model elements do not affect the matching outcome. For instance, in the parameter injection adaptation case, it is meaningful to match the input parameters of the BWS operation against the output parameters of the CWS. This indicates that the participating elements are limited to: *Part*, *InMessage*, *OutMessage* and *ValueObject*. The rest of the modeling elements (e.g., *BusinessService* or *ContextService*) are irrelevant to the matching and are, therefore, not present in the PCG.

The set of PCG graphs for the parameter injection adaptation are depicted in Fig. 5. The names in the nodes are presented in the form <model name>! <model element> with model name B referring to the BWS model and model name C referring to the CWS model. The first graph captures the similarity that should exist between the input parameter of the BWS operation (*B!Part*) and the output parameter of the CWS operation (*C!Part*). The remaining nodes capture the relation of this node to the other model elements. The second graph captures the similarity between the input parameter of the BWS operation (*B!Part*) and the name of the *WSOperation* of the CWS. It is desirable to perform the matching also on this level, since it is usual for Web Service operations to have an expressive name depicting the kind of information they provide (e.g., *getCountryByIP*). The edge names have been created by concatenating the respective names in the BWS and the CWS model. This is a neces-

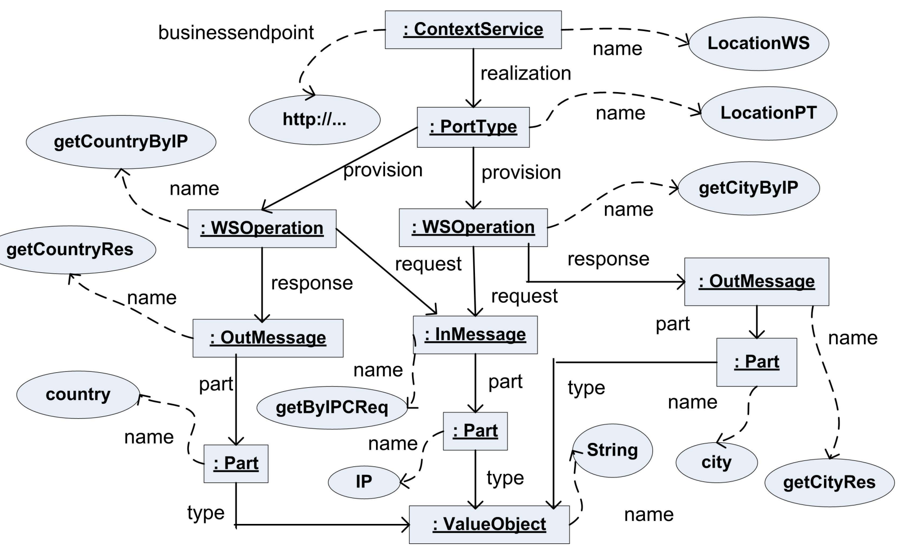

Figure 3: Typed attributed graph of *LocationWS* Context Web Service.

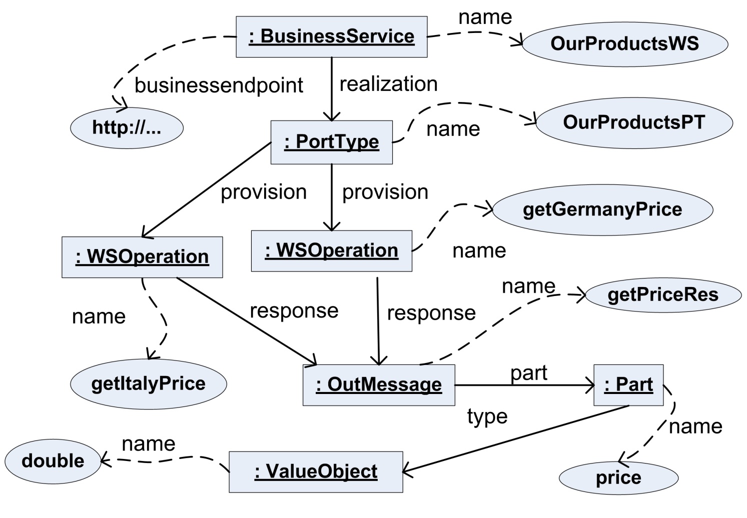

Figure 4: Typed attributed graph of *OurProductsWS* Business Web Service.

sary preprocessing step for applying the similarity flooding algorithm that constructs the PCG based on identical edge names in the input graphs.

The graphs for operation selection are presented in Fig. 6. The matching of this category is applicable on BWSs that expose operations with a high degree of similarity, i.e., operations that perform similar processing offering the same functionality on different parameters or in a different way. Therefore, the PCG is constructed only for model graphs that adhere to this prerequisite. The graph elements present in the PCGs are the name of the BWS operation (*B!WSOperation*) that needs to match the output parameter of the CWS operations (*C!Part*). As in the parameter injection case the name of the BWS operation participates in the matching. This is expressed in the second graph of the set, where the operation names are matched (*B!WSOperation* and *C!WSOperation*). More details on the necessity of the matching rules can be found in a previous publication of the authors [6].

2.3 Flooding on Graphs

The element similarity (on the graph nodes) is calculated based on a similarity function that can employ different matching algorithms or tools for syntactic or semantic matching (e.g., Levenshtein distance, custom matching algorithms). The structural similarity is addressed by the subsequent calculations of the similarity function in the similarity flooding algorithm:

$$\sigma_n^{i+1}(n_x, n_y) = \alpha \times \sigma_n^i + \beta \times \sum_{m \in edges(n)} w(m, n) \times \sigma_m^i(m_x, m_y)$$

The similarity between the nodes is computed based on the similarity of the nodes in the previous iteration of the algorithm and the similarity degree of the adjacent nodes in the PCG. The constants α and β depict the importance of the initial similarity value, which is based purely on the matching on the labels of the nodes, and the importance of the node neighbors respectively. This follows the same concept for weighting the linguistic or semantic versus the

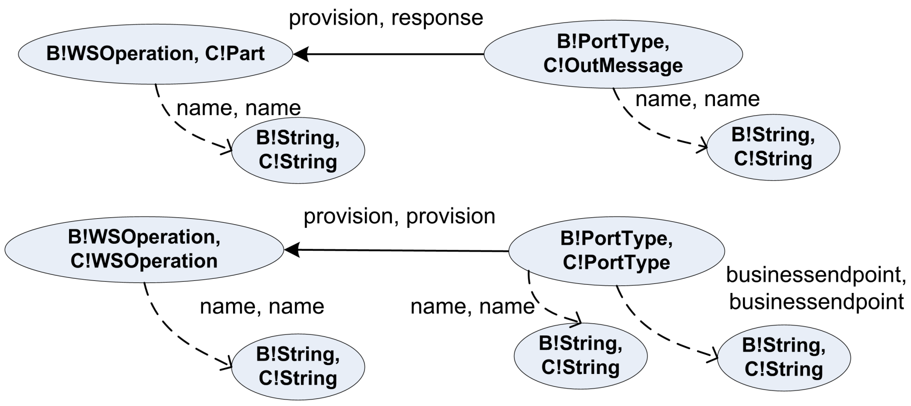

Figure 5: Pairwise connectivity graphs for context matching for *parameter injection*.

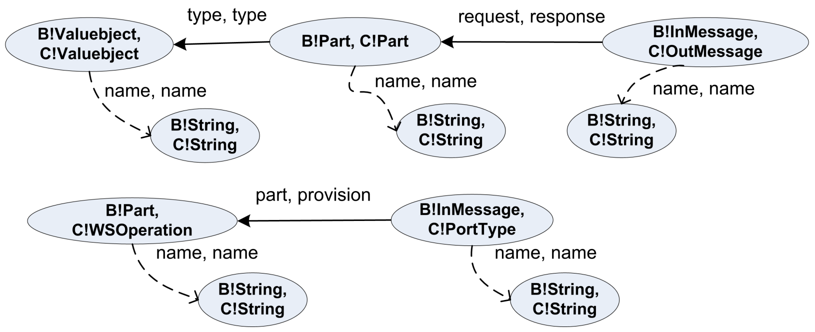

Figure 6: Pairwise connectivity graphs for context matching for *operation selection*.

structural similarity of two elements as in [23]. The importance of structural similarity is not the same as in generic model matching or comparison approaches that capture the structure of the whole model. In the current problem the relation with other model nodes is known in advance and is limited to a subset of the model graph as expressed in the respective compatibility graphs. As in the original similarity flooding algorithm in each iteration the similarity values for each node are normalized in order to result in $\sigma_n^i \in (0,1)$ by dividing each value with the maximum value obtained in that iteration.

In the original similarity flooding algorithm the iterations are performed until the similarity values in each iteration differ slightly from the previous ones, i.e., the delta becomes less than a threshold ϵ. This is not applicable in the addressed case due to the small size of the PCGs; the algorithm with the consecutive calculations of the similarity function is terminated when all graph nodes have been reached.

3. SIMILARITY FUNCTIONS

The similarity outcome is dependent on the initial algorithm employed for the computation of σ_n^0. Various algorithms for calculating the similarity distance can be considered [2] with all providing values of $\sigma_n^0 \in (0,1)$.Three different metrics are used in this work: 1) Levenstein distance, 2) WordNet lexical database [19], and 3) Trigrams. As aforementioned in contrast to conventional approaches two elements to be matched may not correspond to the same metamodel element: for instance, a parameter name may be compared with a class name. This is necessary in the current problem in order to capture the context adaptation cases. Nevertheless, in the typed attributed graph representation of the previous Sections all model elements are treated in a uniform manner.

3.1 Levenshtein distance

The Levenshtein distance normalized to the length of the element names being compared is used as described in [5]:

$$\sigma_n^0(n_x, n_y) = 1 - lev(n_x, n_y)/max(length(n_x), length(n_y))$$

For instance, if the Levenshtein distance is applied for the first graph of the parameter injection adaptation case on elements *B!Part* with name *cityName* and *C!Part* with name *city*, the computation will be as following:

$$\sigma_n^0(cityName, city) = 1{-}0.5/max(8,4) = 1{-}0,0625 = 0,938$$

This example is displayed in Fig. 7. where the propagation coefficient range has been assigned to 1.0 for all edges in PCG. Generally, other computations can be assumed giving a result with value from 0.0 to 1.0. By applying the similarity flooding algorithm considering the initial similarities of the neighbors: $\sigma_m^0(String, String) = 0,833$ and $\sigma_k^0(getTempReq, getCityRes) = 0,95$ the outcome of the next iteration, with constants α and β both set to 1.0, becomes:

$$\sigma_n^1(cityName, city) = 0,938{+}1.0{\times}0,833{+}1.0{\times}0,95 = 2,721$$

, which is normalized to 1.0 as the highest value of the first iteration.

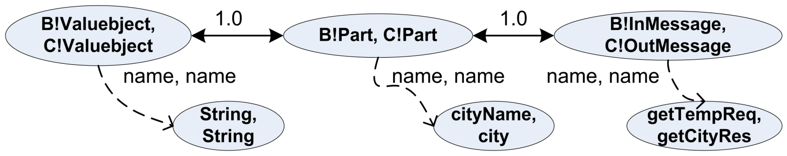

Figure 7: Propagation graph for context matching for *operation selection.*

3.2 Wordnet lexical database

Wordnet is a lexical database that retrieves similar concepts to the input word given [19]. The introduced Wordnet distance considers same words (same), synonyms (syn), meronyms (mer), hypernyms (hyper) and related terms (rel):

$$\sigma_n^0(n_x,n_y) = l\times\sigma_{same}+p\times\sigma_{syn}+q\times\sigma_{mer}+r\times\sigma_{hyper}+t\times\sigma_{rel}$$

, where the constants l, p, q, r and t express the importance of each similarity level retrieved through Wordnet. At most one of the operands in the similarity calculation will evaluate to 1. Some justified values to give priority to synonym matching would be: $l = 1.0$, $p = 0.9$, $q = r = 0.6$, $t = 0.05$.

Composite names consisting of more than one words may appear in WS models (e.g., *getTemperature*). These composite terms need to be taken into account in the similarity calculation in the semantic comparison using Wordnet. Thus, the similarity value is calculated as the average $avg(\sigma_n)$ considering the similarity calculation of the meaningful tokens (substrings).

3.3 Similarity based on n-grams

N-grams refer to a contiguous sequence of n items from a given sequence of text or speech [20]. For the similarity algorithm trigrams have been considered: the count of equal character sequences of size three is compared to the total number of sequences of three characters (including spaces added during the tokenization):

$$\sigma_n^0(n_x,n_y) = trigram(n_x,n_y)$$

4. EVALUATION DEMONSTRATION

In order to demonstrate the proposed approach the example BWS models (i.e., *TemperatureWS* and *OurProductsWS*) along with *HotelsService*[3] are being compared with *LocationWS* along with additional models retrieved from the seekda Web Service search engine:

- *IP2Geo*[4]
- *Geolizer*[5]
- *SIGeoLocation*[6]
- *Ip2LocationWebService*[7]
- *IpToLocationWS*[8]
- *UKLocation*[9]
- *GeoIPService*[10]
- *IP2CountrySe-rvice*[11]
- *Ip2countryws*[12] and
- *LocationByZipService*[13]

The above services provide an evaluation set of 33 comparisons. By looking at the WSDL specifications, it is observed that *LocationWS*, *IP2Geo*, *IPSIGeoLocation* and *Ip2LocationWebService* match *TemperatureWS*, whereas those and additionally *GeoIPService*, *IpToLocationWS*, *IP2CountryService* and *IP2CountryWS* match *OurProductsWS* and *HotelsService*, since they provide location information as required by the BWSs.

The compatibility graphs for the WS comparisons were constructed based on the WSDL descriptions available in seekda. These were then used as input for the modified version of the similarity flooding algorithms that employed the existing implementation of the similarity flooding algorithm[14]. The elements participating in the compatibility graphs are matched on the basis of the similarity functions presented in the previous Section. For convenience and algorithm application purposes the label of each node corresponds to the value of the name type node. The employed Web Service descriptions contained in many cases *complex type* elements defined in the XML Schema at the beginning of the WSDL specification. These complex types were broken down to their ingredient elements and were added in the corresponding model graphs as such in order to be used as input.

From the part of the first BWS, i.e., *TemperatureWS*, one operation was available, i.e., *getTemperature*, but this was not the case for the CWSs that provided more than one operations. In the executions the maximum similarity value among the CWS operations was considered. For the *OurProductsWS* the similarity flooding algorithm variant has been applied on both operations and the average values of the results have been used for the diagram construction, whereas from *HotelsService* the *SearchHotels* operation was considered. Note that for conformity purposes in services offering multiple bindings the SOAP binding was used. For all cases similarity values higher than the threshold of 0.5 have been treated as successful matches. The results were calculated in the form of the following three metrics widely used in the information retrieval field [5, 23]:

$$precision = \frac{\#correctServicesReturned}{\#totalCorrectServices}$$

[3]http://demo.touricoholidays.com/WS/HotelsService.asmx
[4]http://ws.cdyne.com/ip2geo/ip2geo.asmx
[5]http://dns02.de/geolizer/server.php
[6]http://ws.strikeiron.com/IPligenceGeoIPLocation?WSDL
[7]http://ws.fraudlabs.com/ip2locationwebservice.asmx
[8]http://www.buysellbusiness.org/ws/IpToLocationWS.asmx
[9]http://www.webservicex.net/uklocation.asmx
[10]http://www.webservicex.net/geoipservice.asmx
[11]http://www.flash-db.com/services/ws/ip2Country.wsdl
[12]http://www.tarya.co.uk/ws/Ip2countryws.asmx
[13]http://www.flash-db.com/services/ws/locationByZip.wsdl
[14]http://infolab.stanford.edu/ melnik/mm/sfa/

$$recall = \frac{\#correctServicesReturned}{\#totalServicesReturned}$$

$$f - measure = \frac{2 \times precision \times recall}{precision + recall}$$

The results are presented in Figs. 8 to 10. The results on the left represent the values, when only the initial similarity values are considered σ_n^0 without the iterations of the similarity flooding algorithm, whereas the results on the right correspond to the final matching results after the algorithm execution. From the results on parameter injection it is observed that the similarity flooding algorithm does not always produce better matching results than the sole application of the initial similarity functions. This is due to the different nature of the structural similarity of the input graphs. Moreover, when a CWS operation response provides a *complexType* element consisting of various atomic elements, the result accuracy is lower due to the dependency with the neighbor nodes. From the diagrams the trigram seems to provide better results. However, the exploitation of Wordnet should be a better choice for this kind of problem, since semantics have an important role in identifying the proximity of the context concepts captured in the WS descriptions. This is an indication of the need to consider the use of hybrid approaches in the similarity functions.

The proposed approach can be inefficient for adaptation cases that are present, but, however, not captured in the service model (or respectively in the elements of the typed attributed graph). In such cases, the addition of service description elements in the model can be regarded in conjunction with ontologies, as proposed in existing works [18].

5. RELATED WORK

As aforementioned model matching is usually applied for software evolution purposes. Existing works on model differentiation detect points of variation between different versions of the same system or in various applications areas, such as behaviors of formal specications and clone instances. An overview and classification of different model differentiation techniques divided into the steps of Calculation, Representation and Visualization can be found in [11].

Some approaches are more generic, whereas others refer to specific modeling languages. UMLDiff [25] finds differences between subsequent versions of Unified Modelling Language (UML) models using reverse engineering to compute the models from the code. It is based on the characteristics of object-oriented systems in order to present the code elements, i.e., packages, classes, interfaces, fields and blocks, and their relationships and then identify the changes performed between two versions of the same model. Differences in UML models are also addressed in [15]. In the industry, where it is usual to model sotware systems in UML, model differentiation is also supported by tools that function like the traditional version control system (e.g., concurrent modeling in Apache Subversion). DSMDiff [13] concentrates on comparing domain specific models instead of using UML notation. It introduces a number of algorithms for detecting model differentiation including signature and structural matching.

Some works follow a theoretical approach, whereas others provide matching environments. Plugins of the Eclipse IDE can be found in EMF Compare and the Epsilon Comparison Language (ECL) of Epsilon. ECL allows for matching between different types of models, but is usually employed with EMF models. The language possesses many degrees of expressiveness allowing the user to define her/his own matching rules using the internal mechanisms provided by the Epsilon Object Language (EOL) or employ external matching defined in an independent programming language. More recent works applied on domain-specific models can be found in GenericDiff [24].

In many of the above approaches, when models are matched they are represented as typed attributed graphs [3], with each node in the graph representing model elements. When graphs are considered the model matching takes into account the relations that exist among its elements as expressed in the graph edges. This is addressed in the similarity flooding algorithm [14], which is employed in the current work. The algorithm can be applied on models of different types; not only graphical models conforming to a metamodel, but also XML schemas, data instances etc. represented as typed attributed graphs. The models are matched on the basis of the similarity flooding algorithm constructed on the fact that the elements of the two models are similar when their adjacent elements are similar. The similarity flooding algorithm has been applied for the matching of metamodels represented as graphs using five configurations with different levels of detail in [5]. The area of schema matching is also addressed in [23], where models are transformed using planar notation and are then matched using an enhanced version of the planar graph edit distance algorithm.

The above works concentrate on general matching and comparison solutions, but cannot be applied on the context dependencies matching problem. The current work has been influenced by the existing literature, such as the similarity flooding algorithm and its different uses, but the matching rules are defined on a different level and applied on a graph forest instead of a sole graph representing the whole model structure.

6. CONCLUSIONS

This work has focused on providing matching actions for Web Service models that follow the notation of the WS metamodel presented as a typed attributed graph. A variant of the similarity flooding algorithm was applied on Business Web Service and Context Web Service models, in order to examine whether the matching of input models on context dependencies is feasible. All models have been represented as typed attributed graphs and have been matched using similarity functions for string matching. The results of the evaluation through a number of constructed and existing Web Services from online service registries demonstrates the matching outcome that can be achieved. This technique can be useful for software engineers that wish to exploit existing WS descriptions in order to build wider applications with context-aware characteristics.

As future work the use of WS models that do not include any information on whether they require or offer context information will be considered. This approach will allow the exploitation of the plethora of WS descriptions available on online service registries. We intend to implement the matching rules analyzed using the Epsilon Comparison Language[15]. An initial evaluation of its use has already been conducted. The use of ECL can be complemented with the

[15]http://www.eclipse.org/epsilon/doc/ecl/

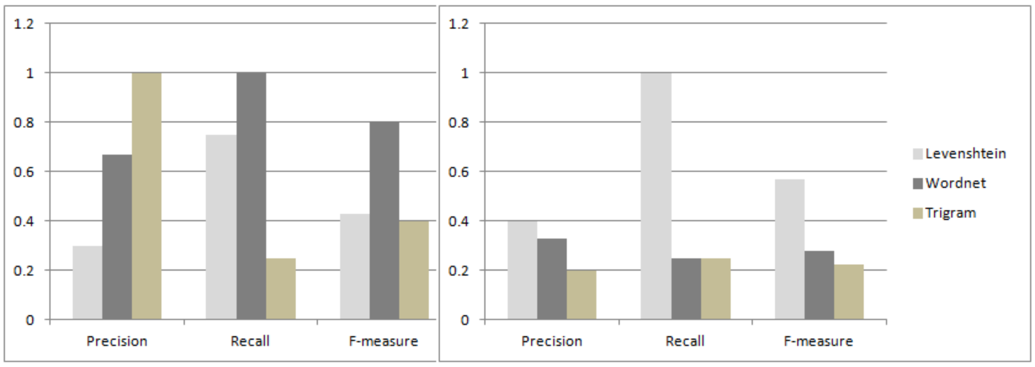

Figure 8: Initial (*left*) and (*right*) results of similarity flooding matching *TemperatureWS* with CWSs.

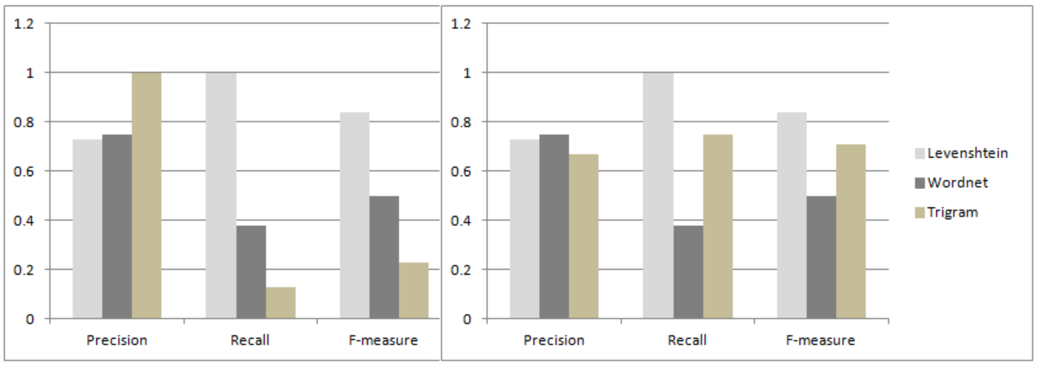

Figure 9: Initial (*left*) and final (*right*) results of similarity flooding matching *OurProductsWS* with CWSs.

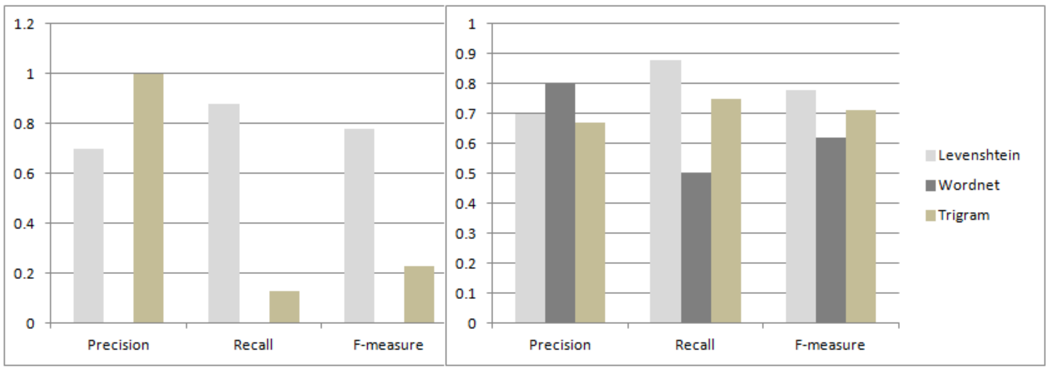

Figure 10: Initial (*left*) and final (*right*) results of similarity flooding matching *HotelsService* with CWSs.

EMF Compare tool to add visualization to the results of the matching. Another area of special interest lies in WS composition in the form of workflows as described in existing specifications, such as the Business Process Execution Language (BPEL). The presented work constitutes a special case of service composition. The study of WS composition possibility through model matching presents an area that can be exploited by software engineers in combination with online service registries.

7. REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, UK, 1999. Springer-Verlag.

[2] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[3] J. de Lara, R. Bardohl, H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Attributed graph transformation with node type inheritance. *Theor. Comput. Sci.*, 376(3):139–163, May 2007.

[4] H. Ehrig, U. Prange, and G. Taentzer. Fundamental theory for typed attributed graph transformation. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *Graph Transformations*,

volume 3256 of *Lecture Notes in Computer Science*, pages 161–177. Springer Berlin Heidelberg, 2004.

[5] J.-R. Falleri, M. Huchard, M. Lafourcade, and C. Nebut. Metamodel matching for automatic model transformation generation. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 326–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] G. M. Kapitsaki. Identifying context sources towards context-aware adapted web services. In *WEBIST*, pages 135–140, 2011.

[7] G. M. Kapitsaki and A. Achilleos. Applying model-driven engineering for linking web service and context models: position paper. In *iiWAS*, pages 511–514, 2011.

[8] G. M. Kapitsaki, D. A. Kateros, G. N. Prezerakos, and I. S. Venieris. Model-driven development of composite context-aware web applications. *Information & Software Technology*, 51(8):1244–1260, 2009.

[9] G. M. Kapitsaki, D. A. Kateros, and I. S. Venieris. Architecture for provision of context-aware web applications based on web services. In *PIMRC*, pages 1–5, 2008.

[10] A. G. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[11] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige. Different models for model matching: An analysis of approaches to support model differencing. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM '09, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Y. Lee, S. S. Iyengar, C. Min, Y. Ju, S. Kang, T. Park, J. Lee, Y. Rhee, and J. Song. Mobicon: a mobile context-monitoring platform. *Commun. ACM*, 55(3):54–65, Mar. 2012.

[13] Y. Lin, J. Gray, and F. Jouault. *DSMDiff: A differentiation tool for domain-specific models*, 2007.

[14] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society.

[15] D. Ohst, M. Welle, and U. Kelter. Differences between versions of uml diagrams. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-11, pages 227–236, New York, NY, USA, 2003. ACM.

[16] Q. Z. Sheng and B. Benatallah. Contextuml: A uml-based modeling language for model-driven development of context-aware web services development. In *Proceedings of the International Conference on Mobile Business*, ICMB '05, pages 206–212, Washington, DC, USA, 2005. IEEE Computer Society.

[17] D. Skogan, R. Gronmo, and I. Solheim. Web service composition in uml. In *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*, EDOC '04, pages 47–57, Washington, DC, USA, 2004. IEEE Computer Society.

[18] S. Staab, T. Walter, G. Gröner, and F. S. Parreiras. Model driven engineering with ontology technologies. In *Reasoning Web*, pages 62–98, 2010.

[19] M. M. Stark and R. F. Riesenfeld. Wordnet: An electronic lexical database. In *Proceedings of 11th Eurographics Workshop on Rendering*. MIT Press, 1998.

[20] C. Y. Suen. n-gram statistics for natural language understanding and text processing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-1(2):164 –172, april 1979.

[21] C. Treude, S. Berlik, S. Wenzel, and U. Kelter. Difference computation of large models. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 295–304, New York, NY, USA, 2007. ACM.

[22] H.-L. Truong and S. Dustdar. *A survey on context-aware web service systems*, 2009.

[23] K. Voigt and T. Heinze. Metamodel matching based on planar graph edit distance. In *Proceedings of the Third international conference on Theory and practice of model transformations*, ICMT'10, pages 245–259, Berlin, Heidelberg, 2010. Springer-Verlag.

[24] Z. Xing. Model comparison with genericdiff. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 135–138, New York, NY, USA, 2010. ACM.

[25] Z. Xing and E. Stroulia. Umldiff: an algorithm for object-oriented design differencing. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 54–65, New York, NY, USA, 2005. ACM.