Semantic API Matching for Automatic Service Composition

Doina Caragea
Dept. of Computer Science
Iowa State University, Ames, Iowa

Tanveer Syeda-Mahmood
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

ABSTRACT

In this paper, we address the problem of matching I/O descriptions of services to enable their automatic service composition. Specifically, we develop a method of semantic schema matching and apply it to the API schemas constituting the I/O descriptions of services. The algorithm assures an optimal match of corresponding entities by obtaining a maximum matching in a bi-partite graph formed from the attributes.

1. INTRODUCTION

With an increasing number of organizations putting their business competencies as a collection of web services, it is conceivable that other users could integrate them to create new value-added services in ways that were not anticipated by their original developers. The assembly of services currently requires considerable manual intervention, including examination of the API descriptions of services, and writing of specific code to chain a sequences of services using attributes in the I/O descriptions of services. Semantic schema matching used earlier for data schemas [4,6] can also be applied to APIs allowing automatic orchestration relieving the burden of programming by end users. It can also enable automatic service composition when augmented with service discovery techniques [1,3,5,8].

In this paper, we focus on the automatic matching of API descriptions of services. This is a difficult problem, in general, as the relationship between attributes may not be obvious from their names, types and structural grouping of attributes. Furthermore, a source attribute may be split across multiple destination attributes & vice versa.

2. SEMANTIC SCHEMA MATCHING

We now formulate the problem of semantic match of the APIs of a set of services. The source and target attributes can be regarded as the nodes of a bi-partite graph. Their correspondence is then a matching in the bipartite graph. If the weights on the edges of the bipartite graph reflect the similarity of the attributes, the optimal matching is a matching of maximum cardinality and maximum weight. The semantic schema matching approach, therefore, is to reflect the knowledge of API variables in determining similarity of attributes and use a conventional maximum matching algorithm. Specifically, we used the algorithm of Goldberg and Kennedy[2]. In this algorithm, the matching is computed by setting up a flow network with weights such that the maximum flow corresponds to a maximum matching [2].

2.1 Computing Similarity between attributes

We derive the similarity between attributes using four sets of cues, namely, (1) lexical name matching, (2) semantic name matching, (3) type matching, and (4) structural matching. Due to space limitations, we will expand on semantic similarity while briefly describing other similarity metrics used. The lexical similarity measure captures the similarity in the spelling of names used in APIs (eg. lname, lastname) and is measured as

$$L(A,B) = 2 * \text{Length}(LCS(A,B))/(\text{Length}(A) + \text{Length}(B)) \quad (1)$$

Where A,B are the attributes, and LCS(A,B) is longest common subsequence of A and B.

Type matching:

For APIs the type of attributes is a strong cue in matching, for without proper type casting, the service cannot be launched. The type similarity measure is given by:

Type(A,B) = 1.0 (lossless conversion), = 0.5 (lossy), = 0 (otherwise) (2)

Where the lossless conversion is determined by navigating the reference type hierarchy of a language (eg. int to float is lossless, while float to int is lossy). The existence of an explicit type casting function (eg. 2d array to 1d vector converter code) is also a case of lossless conversion.

Structural matching:

Using the rationale that the grouping of attributes under a node of a particular height in the API schema denotes a concept abstraction used by programmers, the structural similarity is measured by

$$\text{Struct}(A,B) = 1 - (|D(A) - D(B)|)/\max\{D(Gi), D(Gj)\} \quad (4)$$

where D(A) and D(B) are the height of the attributes in their respective schema trees Gi and Gj and D(Gi) and D(Gj) are the heights of the trees

Semantic Name Similarity:

The semantic name similarity is computed using a technique similar to the one in [4], in that, we parse the words to extract tokens and find ontological similarity in the tokens. The parsing uses tokenization, part-of-speech tagging, filtering and abbreviation expansion to generate list of candidate words. Thus CustomerPurchase will be separated into Customer and Purchase. The tokenization uses font changes, underscores, spaces and other separating characters. Abbreviations such as CustPurch will be expanded into CustomerPurchase, CustomaryPurchase, etc, using a domain-dependent abbreviation expansion dictionary generated a priori. Filtering removes stop words and part-of-speech tagging classified words as nouns, adjectives, etc. The resulting words are then used to index an ontology (we use Wordnet Thesaurus[7]) to obtain a list of synonyms. Let each pair of source and destination attributes (A,B) have m and n valid tokens and let Si and Sj be their expanded lists based on ontological processing. We consider each token i in source attribute A to match a token j in destination attribute B if $i \varepsilon Si$ or $j \varepsilon Si$. The candidate matches again form a small bipartite graph in which each edge has flow of unit 1 (Note this graph is different from the API match graph described earlier). The maximum cardinality matching in this graph then denotes the best set of matching word tokens. The semantic name similarity measure is then given as

$$\text{Sem}(A,B) = 2*\text{MaxMatch}(A,B)/(m + n) \quad (4)$$

The semantic similarity allows us to match attributes such as (state,province), (CustomerIdentification, ClientID), (CustomerClass, ClientCategory), etc.

Th overall cost function for computing the edge cost is then

$C(A,B) = \alpha1 * L(A,B) + 2 * Sem(A,B) + \alpha3 * Type(A,B) + \alpha4 * Struct(A,B)$ (5)

The weights (α1,2,3,4) are chosen to be step functions with value = 1.0 until a lower threshold is reached. Thus α1=0.0 for L(A,B) < 0.9 (since lexical similarity is a good indication of relationship for only high-scoring matches), and α2 = 0.0 for Sem(A,B) < 0.67, α3 = 0.0 for Type(A,B) < 0.5, α4 = 0.0 for Struct(A,B) < 0.5. The thresholds were derived by computing the similarity per cue for the actual mapping indicated by programmers for integration of candidates services used for testing and taking their average value.

3. RESULTS

We tested the performance of semantic schema matching on 240 distinct pairs of web services drawn from Crossworlds business object library. The business objects tend to have a larger number of member attributes (over 100), so that the algorithm performance could be gauged on large schemas. We then measured the performance by comparing to a manual match of the attributes of the respective schemas. The number of spurious (false positives) as well as missing matches (false negative), were noted in each pair-wise match.

Table 1 illustrates the matching similarly, for a pair of ADTs depicted in Figure 1. Here a web service that provides a description of an inventory item is chained with a web service that retrieves vendor information associated with the inventory item. A match of InventoryType and StockType has been aided by semantic name matching, while abbreviation expansion has allowed match of InvLocationID to InventoryLocationID

Representative performance for a sampling of web services is illustrated in Table 2. Overall, the system erred on the side of making false positives and was able to maintain a matching accuracy in the range of 75-85%.

4. CONCLUSIONS

In this paper, we have presented an approach to semantically match two API schemas to enable the chaining of their associated services. Building automation into this task enables scalable deployable solutions in the world of internet where the web services are being added at a rapid pace.

4. REFERENCES

[1] T. Berners-Lee et al. The semantic web. Scientific American, 2001.

[2] A. Goldberg and Kennedy. An efficient cost-scaling algorithm for the assignment problem. SIAM Journal on Discrete Mathematics, 6(3):443{459, 1993.

[3] J. Blythe et al. The role of planning in grid computing. In Proc. ICAPS, 2003.

[4] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In The VLDB Journal, pages 49-58, 2001.

[5] D. McDermott. Estimated regression planning for interactions with web services. In Proc. AIPS, 2002.

[6] S. Melnik et al. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In Proc. ICDE, 2002

[7] G.A. Miller. Wordnet: A lexical database for English. Communications of the ACM, 38(11):39{41, 1995.

[8] Evren Sirin, James Hendler, and Bijan Parsia. Semi automatic composition of web services using semantic descriptions.

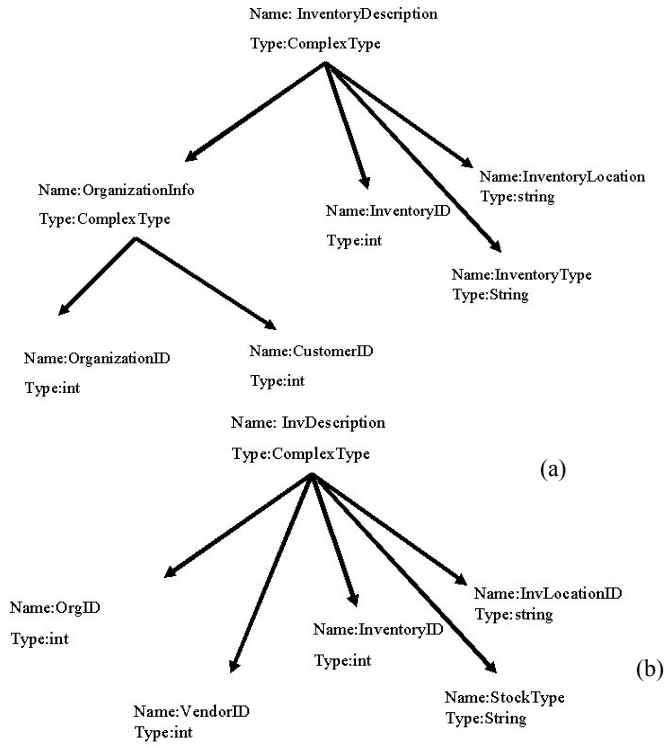

Figure 2: Illustration of semantic matching between APIs for web services exchanging business objects.

S. No.	Source attribute	Matching destination attribute	Matching Score	Contributions in order
1.	OrganizationID	OrgID	2.5	0.67,1.0,1.0, 0.50
2.	InventoryLocation	InvLocationID	3.0	0.74,0.67,1.0,1.0
3.	InventoryID	InventoryID	4.0	1.0,1.0,1.0, 1.0
4.	InventoryType	StockType	3.0	0.56,1.0,1.0, 1.0

Table 1: Matches produced by semantic match for the pair of services ADTs shown in Figure 1.

S.No	Source ttributes	Destination attributes	Correctly matched	Missed matches	Spurious matches	Actual matces	% Accuracy
1.	10	15	8	1	2	9	81%
2.	23	34	28	3	7	31	81.57%
3.	67	73	29	5	9	34	79%
4.	84	56	10	3	4	13	76.4%

Table 2: Illustration of performance of semantic matching during chaining of services derived from business objects.