Towards Ontology Matching for Intelligent Gadgets

Oszkar Ambrus, Knud Möller, and Siegfried Handschuh

Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway (NUIG)
{oszkar.ambrus,knud.moeller,siegfried.handschuh}@deri.org

Abstract. The FAST gadget development environment allows users to graphically compose intelligent, i.e., semantically annotated gadgets from predefined building blocks and deploy them on various mashup platforms, thus enabling the interconnection of different systems and services. In an environment where different parties use different ontologies to describe such building blocks, ontology matching is crucial. This paper discusses first steps in our effort to integrate ontology matching in an end-user-oriented environment such as FAST. We evaluate a number of tools and approaches for solving different levels of complexity in ontology matching and define the direction of integrating ontology matching into FAST.

Keywords: ontology matching, end-user, mashups, gadgets, widgets.

1 Introduction

FAST (Fast and Advanced Storyboard Tools) [1] is a visual programming platform allowing business users to build enterprise-class mashups, employing various underlying services and generating new ones. Resources in FAST are described semantically using different ontologies and vocabularies, and can therefore be combined to what we can call "intelligent gadgets". These ontologies come from different parties, but will sometimes cover the same domain, making ontology matching necessary. The paper reports on early steps on how to integrate existing ontology matching approaches into an end-user-targeted tool such as the FAST platform. The focus of the paper is not on new methods and algorithms for ontology matching, but rather a survey and application of existing ones to our use case of ontology-matching for end-users. We will present automated solutions for simple cases as well as identify more problematic cases which require manual work, in which we want to support the ontology engineer.

In the remainder of this section we give an introduction to the FAST project, and a brief overview of ontology matching. In Sect. 2 we detail the requirements for ontology matching in FAST. Sect. 3 describes the alignment tool used and the rationale behind our choice. In Sect. 4 we describe a problem scenario that we want to give a solution for in FAST, which is the basis of this work, and in Sect. 5 we present the ontologies used for the different web services involved in the scenario. Finally, in Sect. 6 we present the testing procedure and the results, based on which we draw the conclusions in Sect. 7 and define future directions.

A. Dan, F. Gittler, and F. Toumani (Eds.): ICSOC/ServiceWave 2009, LNCS 6275, pp. 570–579, 2010.

1.1 FAST

The main goal of the FAST Project [1] is to develop a web-based user-centric visual programming environment allowing users to build enterprise mashups [2] using so-called *gadgets* (see Fig. 1 for a screenshot exemplifying the composition of so-called *screens* to build such a gadget). The motivation behind FAST is to allow non-technical users to be involved in the development process of software applications based on their ad-hoc needs.

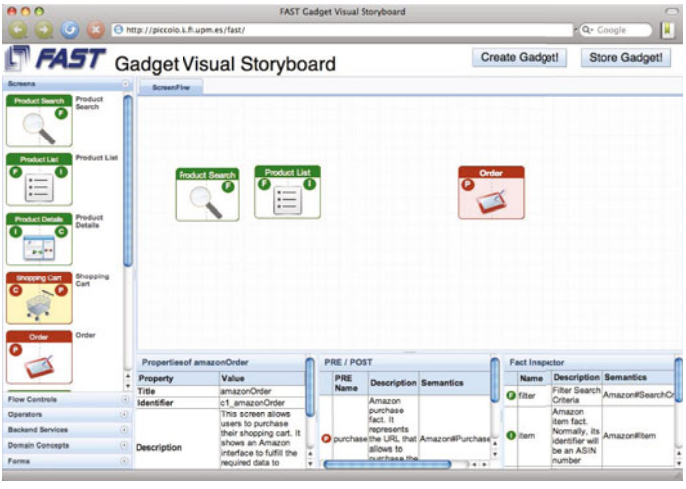

Fig. 1. Composing an intelligent gadget from screens in FAST

The relevant components of the enterprise mashup paradigm are *resources*, which are in the focus of this paper, representing all building-blocks of gadgets (such as the screens in Fig. 1), *gadgets* that provide a graphical interface and interaction mechanisms abstracting from the complexity of the underlying backend, and, *mashups* created by users using the provided gadgets.

1.2 Ontology Matching

FAST uses ontologies to conceptualise the underlying resources used by the different components. Ontologies embody the fundamental vehicle for conceptualising data on semantic systems; they describe the context and semantic background of data that should be known to all agents using it [3]. However, different ontologies are often used to describe the same domain or cover the same scenario. This is also true for FAST, where gadget building blocks can originate from different providers, who might use different ontologies to describe them, so the task of ontology matching is critical for interoperability.

Given two ontologies O and O' that need to be mapped to each other, we adopt the definition given in [4]: an ontology mapping element is a 5-tuple

$< id, e, e', n, R >$, where id is a unique identifier, identifying the mapping element, e and e' are entities (formulas, terms, classes, individuals) of the first and second ontology, respectively, n is a confidence measure holding the correspondence value between e and e', R is the correspondence relation holding between e and e' (e.g., `equivalence (=)` or `more general(`$\sqsupseteq$`)`). The alignment operation determines the mapping M' for a pair of ontologies O and O'. The alignment process can be extended by parameters, such as an input mapping, weights and thresholds and other external resources (dictionaries, thesauri, etc.). Different levels of mappings are defined: (a) A *level 0* mapping [5] is a set of the above mapping elements, when the entities are discreet (defined by URIs). See List. 2 for an example. (b) A *level 1* mapping is a slight refinement of level 0, replacing pairs of elements with pairs of sets of elements. (c) A *level 2* mapping can be more complex and defines correspondences in first order logic. It uses the ontology mapping language described in [6]. It can describe complex correspondences, such as the one detailed in Sect. 6.2.

2 Ontology Matching in FAST

The gadget life cycle in FAST has several phases and roles associated, as detailed in [1]. Here, we list the ones relevant for the ontology matching tasks, in decreasing order of the measure in which knowledge about ontologies is required. Note that several roles can be played by the same actor. (i) The *ontology engineer* creates the ontologies used to annotate services and data. This role also includes the process of ontology matching, either automated or manually, determining if the alignment is feasible and creating so-called *matching operator* building blocks, which are basic elements of the FAST screen building. The *resource developer* then uses these ontologies to annotate resources created in FAST. (ii) Ontology matching is needed by the *screen developer* at the design-time of a screen (a visual building block of a gadget). Screen developers have a dedicated UI component for building screens, in which they can use the matching operators to combine components annotated with different ontologies. No actual matching needs to be performed in this phase, but rather the possibility of matching needs to be determined (i.e., *can* two screens A and B be combined?). (iii) The *gadget developer* combines screens to screen-flows and gadgets, and only uses ontology matching implicitly. (iv) The *end-user* uses the final deployed gadget at run-time, but is unaware of the underlying resources and ontologies or the matching process. Only at run-time the actual mapping of instance data has to be performed. In this paper, we mainly consider the first two cases (i.e. ontology engineering and screen development).

3 Alignment Tool

An *ontology mapping* is a declarative specification of the semantic overlap between two ontologies [7], being represented as a set of axioms in a *mapping language*. It is the result of the *ontology alignment process* having has three

main phases: (1) discovering the mapping, (2) representing the mapping and (3) exploiting the mapping. Thus we need a tool to assist the ontology engineer in the alignment process. Based on the description given in Sect. 2 we identify the following requirements for an alignment tool, that we will take as the basis for an ontology matching component in FAST: (i) all three phases of the process need to be accessible, (ii) matching of OWL and RDFS ontologies must be supported, (iii) the tool should perform the alignment process with little or no user interference, since FAST is end-user oriented, (iv) the tool needs to be open source, allowing it to be integrated into the free and open FAST platform and (v) it should be well documented.

Based on these requirements, we compared three different tools. *MAFRA* [8] and *RDFT* [9] were found to be unsuitable for either being to restricted to their environment or being no longer available. *Alignment API* [5] is the tool best matching our requirements, satisfying all the desired conditions. It is still under active development, provides an API and its implementation, is open source (GPLv2 or above) and written in Java, providing an easy way to embed it into other programs. Alignment API can be extended by other representations and matching algorithms, it can be invoked through the command line interface (thus working without user interference) or one of the two available GUI implementations, or it can be exposed as an HTTP server. The tool allows for testing different alignment methods and can generate evaluation results based on a reference alignment. Alignment API can generate the mapping results in XSLT, therefore providing an easy way to integrate them into other systems.

4 Scenario Description

In our evaluation scenario, which is taken from the e-commerce domain, a user needs to build a gadget which combines data from major e-commerce services, allowing to aggregate item lists from all of them in a combined interface. As examples in our scenario, we consider the two most popular online shopping websites[1], Amazon and eBay, along with the BestBuy site. The latter is an interesting case, because it exposes its data in RDF using the GoodRelations (GR) ontology [10], which has recently gained a lot of popularity. It is therefore one of the first major e-commerce sites to provide semantic metadata.

Figure 2 illustrates our scenario. There are three retrieval components that wrap the different e-commerce sites and provide data according to three different ontologies: the GR ontology, the Amazon ontology (A) and the eBay ontology (E). Another component displays GR items for display to the user, but not A or E items. If the gadget designer wants to aggregate data from all three services in the display, there will therefore have to be a mapping present between A and E on the one hand, and GR on the other.

[1] `http://alexa.com/topsites/category/Top/Shopping`, checked 01/11/2009

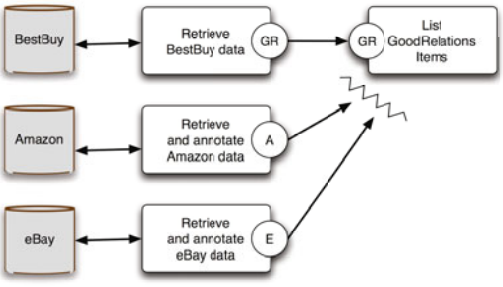

Fig. 2. Three gadget components retrieving incompatible data

5 Ontologies

Of the three ontologies used in our evaluation, only GoodRelations is a real-world, extensive ontology for e-commerce. The other two, i.e., the Amazon and the eBay ontologies were developed for simulation purposes as simplified versions of what would be used in the real-life scenarios. They were designed to showcase particular features of ontology mapping in our scenario.

GoodRelations: This ontology is aimed at annotating so-called "offerings" on the Web, which can be products or services. The ontology features support for ranges of units, measurements, currencies, shipping and payments, common business functions (sell, lease, etc.) and international standards (e.g. ISO 4217) and codes (e.g., EAN) in the field. The main class is `Offering`, which provides a `ProductOrService` with a given `BusinessFunction`. It may be constrained in terms of eligible business partner, countries, quantities, and other properties. It is also described by a given `PriceSpecification`. The super-class for all classes describing products or service types is `ProductOrService`, which is described by its title and description, manufacturer, make and model, etc.

Amazon Ontology: We have created a small Amazon ontology based on a subset of the datatypes supported by the web service exposed by Amazon to third-party agents. The ontology describes `Items` based on the `ItemAttributes` description given in the Amazon Product Advertising API documentation[2]. The ontology features three classes for describing a product. Example instance data is given in List. 1. (1) `Item` represents an Amazon item, defined by a title, a manufacturer, a product group (DVD, Book, etc.), an international EAN code, an ASIN (unique Amazon id), an author (for books) and a `ListPrice`. (2) `Company`, described by a legal name, is used for representing the manufacturer of an `Item`. (3) `ListPrice` has two properties: `hasCurrencyCode`, representing an ISO 4217 currency code (e.g. GBP or EUR), and `hasAmount` representing the price in the given currency.

[2] `http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/`

```
:Item_7590645 a amzn:Item;
   amzn:hasASIN "B0012YA85A";
   amzn:hasManufacturer :Manufacturer_Canon;
   amzn:hasModel "XSI Kit";
   amzn:hasPrice :Price_7590645_1;
   amzn:hasProductGroup "Electronics";
   amzn:hasTitle "Canon Digital Rebel XSi [...]" .
:Manufacturer_Canon a amzn:Company;
   amzn:hasLegalName "Canon" .
:Price_7590645_1 a amzn:ListPrice;
   amzn:hasAmount "575.55";
   amzn:hasCurrencyCode "GBP" .
```

Listing 1. Simplified Amazon ontology data in N3 notation

eBay Ontology: The eBay ontology was created based on the eBay Shopping API[3] and is supposed to annotate data retrieved through the web service described by the API. The ontology features three basic classes, (1) `SimpleItem` represents an eBay `Item`, that is sold by a `SimpleUser`. It is described by a title, a `CurrentPrice` (the highest bid or the fixed selling price), primary category name, manufacturer, model, EAN code, item ID (a unique eBay ID), bid count, end time of bid, country where the item is located, and a product ID (supporting major international product codes — based on the Finding API). (2) The `CurrentPrice` features a `hasAmountType` property (currency code), and a `hasAmount` property (amount of money for a price per unit). (3) `SimpleUser` contains information about eBay users, described by a user ID, about me URL and the seller's positive feedback score. This class will not be used for capturing information on goods for our scenario, but is an essential component of the eBay system, which was the reason for its inclusion in the ontology.

6 Testing and Results

We present the approach an ontology engineer has to take to discover and represent ontology mappings, and a means to exploit them after the they have been discovered and appropriately represented.

There is a major paradigm difference between the GR ontology and the other two ontologies (see Sect. 6.2 for details). After some initial testing, we concluded that automatic mapping from GR to A/E using the string-based methods employed by the Alignment API tool was not feasible. Therefore, the following sections report on *automatic mapping* for the A–E pair — which are similar enough to be suitable for level 0 mapping —, and *manual mapping* for the GR–A/E pairs.

6.1 Automatic Mapping

For automatic finding of level 0 mappings, we used a simple string distance-based algorithm provided by Alignment API [5], which computes the string distance

[3] `http://developer.ebay.com/DevZone/shopping/docs/CallRef/index.html`

between the names of the entities to find correspondences between them. Four methods have been used for computing the distance: (1) equality, which tests whether the names are identical, (2) Levenshtein distance (number of character operations needed), (3) SMOA distance (which is a specialised distance for matching ontology identifiers) and (4) a Wordnet-based [11] distance using the JWNL library with Wordnet.

The alignment description derived from these methods is given based on a simple vocabulary, containing a pair of ontologies and a set of correspondences, which express relations between entities of the two ontologies. We used the level 0 mapping representation for representing simple mappings, which map discrete entities of the two ontologies. Thus the representation of the correspondences is given with the five elements described (with the `id` being optional), as shown in List. 2. Similar mappings were were also used for more complex, manually-created representations (level 2), as detailed in Sect. 6.2.

```
<level_0_mapping> a align:Cell;
   align:entity1 amzn:hasCurrencyCode;
   align:entity2 ebay:hasAmountType;
   align:measure "1.0"^^xsd:float;
   align:relation "=" .
```

Listing 2. Level 0 mapping element example

Testing Procedure: For the Amazon–eBay pair we set up a reference alignment, against which the results are evaluated. We then ran the matching process for all for methods: (1) equality, (2) Levenshtein distance with a confidence threshold of `0.33` (meaning that any correspondence having a smaller confidence measure will be excluded), (3) SMOA distance with a threshold of `0.5` and (4) Wordnet distance using a threshold of `0.5`. To apply the results, we rendered an XSLT template to transform an example dataset.

Results: The results of automatically aligning the Amazon and eBay ontologies were quite favourable. As shown in Tab. 1, we captured the four main parameters used in information retrieval, as described in [12]. These four parameters are used for evaluating the performance of the alignment methods: (1) *Precision*, the fraction of results that are correct — the higher, the better, (2) *Recall*, the ratio of the correct results to the total number of correct correspondences — the higher, the better, (3) *Fallout*, the fraction of incorrect results - the lower the better, and (4) *F-measure*, which measures the overall effectiveness of the retrieval by a harmonic mean of precision and recall — the higher, the better. The first row (reference) shows the reference alignment, which, naturally, has both perfect precision and recall. We can observe what intuition has predicted, namely that pure string equality (equality) is far too simple and irrelevant, by only taking identical labels. By using string distances and giving certain thresholds (Levenshtein and SMOA), we can see that the results are much less precise, but have a better recall, since this allows for entities having similar names to be discovered, at the expense of having quite a few incorrect results

Table 1. Alignment results: Precision, Recall, Fallout and F-Measure

	precision	recall	fallout	f-measure
reference	1.00	1.00	0.00	1.00
equality	1.00	0.38	0.00	0.55
SMOA	0.43	0.75	0.57	0.55
Levenshtein	0.40	0.75	0.60	0.52
JWNL	0.67	0.75	0.33	0.71

(lower precision); the thresholds allow for low-scored cases to be eliminated, although this results in the exclusion of some correct correspondences. The last column (JWNL) contains the results of the Wordnet-enabled method, which shows quite an improvement (precision of 0.67 and a recall of 0.75), due to the lexical analysis, which performs a much more relevant comparison of strings, giving a high number of correct results. The precision of the JWNL alignment shows only a tiny drop below the recall value, meaning that the number of incorrect correspondences discovered is small, and the main source of error is from the number of correspondences not discovered.

We can deduce that the results provided are satisfactory, even though the methods used were simple, string-based ones, and the process was completely automated without any user input. We are therefore confident that through some user assistance or an initial input alignment the tool can achieve 100% correct results.

6.2 Manual Mapping

The GoodRelations (GR) ontology employs a unique paradigm, different from the paradigms of Amazon (A) and eBay (E). In GR everything is centred around an instance of `Offering` and a graph of other instances attached to it, whereas for A (and similarly for E), the main class is `Item`, which holds all relevant properties. In principle, `Item` would correspond to `ProductOrService` in GR, but the properties of the `Item` class are reflected as properties of many different classes in GR.

Though the infeasibility of automating this alignment became obvious, we have represented the alignment in the mapping language supported by the tool, as a level 2 mapping (described in Sect. 1.2). This mapping description can later be used by the run-time gadget code. List. 3 shows an example mapping between two properties of the two ontologies, specifying that the relationship is `Equivalence` with a certainty degree of `1.0`. This fragment does not show, but assumes the equivalence correspondence between the classes `Item` and `Offering`, which is a trivial level 0 mapping. This mapping specifies the relation

$$\forall v, z; hasEAN(v, z) \Longrightarrow \exists x, y; includesObject(v, x) \wedge typeOfGood(x, y) \wedge hasEAN\_UCC\_13(y, z),$$

```
<level_2_mapping> a align:Cell;
    align:entity1 amzn:hasEAN;
    align:entity2
       [ a align:Property;
         align:first gr:includesObject;
         align:next gr:hasEAN_UCC_13,
            gr:typeOfGood ];
      align:measure "1.0"^^xsd:float;
      align:relation "Equivalence" .

amzn:hasEAN a align:Property .
gr:hasEAN_UCC_13 a align:Property .
gr:includesObject a align:Relation .
gr:typeOfGood a align:Relation .
```

Listing 3. Fragment of the Amazon–GoodRelations mapping

meaning that the `hasEAN` property of `v` in the A ontology corresponds to the `hasEAN_UCC_13` property of the `typeOfGood` of the `includesObject` of `v` in GR. The domains and ranges of the properties are inferred, thus it is deduced, that in A `v` is of type `Item` and `z` is `int`, and in GR `v, x, y` and `z` are instances of the classes `Offering`, `TypeAndQuantityNode`, `ProductOrService` and `int`, respectively.

Using this representation, complex correspondences can be modelled, using first order logic constructs.

7 Conclusion

The goal of this paper was to explore ontology matching in an environment such as the FAST platform, where building blocks described with different ontologies need to be integrated by end users. To this end, we have identified which roles in the FAST gadget development lifecycle need to be considered, and which kinds of ontology matching problems they might face. Based on an example scenario from the e-commerce domain, which includes real-world ontologies such as GoodRelations, we have evaluated different existing algorithms for level 0 matching problems. A wordnet-based approach for string matching performed best, giving results suitable for semi-automatic level 0 matching, thus enabling non-expert end users to perform such tasks in FAST. For more complex level 2 matching problems, manual definition of matching rules is still necessary. Additionally, we have evaluated three ontology matching tools based on the requirements given by the FAST platform, and established that the Alignment API tool suits our needs best. We use Alignment API both for the (semi-)automatic generation of level 0 matching rules, as well as for the syntactic representation of manually generated level 2 problems. Based on this format, ontology matching rules of all levels can be represented and executed in all relevant components of the FAST architecture.

As future work, we need to evaluate existing or develop new methods to aid users in complex ontology matching problems (level 2). Additionally, we will

integrate ontology matching into the running FAST platform, and evaluate its performance and usability there.

Acknowledgments

The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2) and (in part) by the European project FAST No. FP7-ICT-2007-1.2 216048.

References

1. Hoyer, V., Janner, T., Delchev, I., Lpez, J., Ortega, S., Fernndez, R., Möller, K., Rivera, I., Reyes, M., Fradinho, M.: The FAST platform: An open and semantically-enriched platform for designing multi-channel and enterprise-class gadgets. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 316–330. Springer, Heidelberg (2009)
2. Hoyer, V., Stanoesvka-Slabeva, K., Janner, T., Schroth, C.: Enterprise mashups: Design principles towards the long tail of user needs. In: SCC 2008: Proceedings of the 2008 IEEE International Conference on Services Computing, pp. 601–602 (2008)
3. Gruber, T.R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In: Guarino, N., Poli, R. (eds.) Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers, The Netherlands (1993)
4. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. Journal on Data Semantics 4, 146–171 (2005)
5. Euzenat, J.: An API for ontology alignment. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 698–712. Springer, Heidelberg (2004)
6. Scharffe, F., de Bruijn, J.: A language to specify mappings between ontologies. In: Proc. of the Internet Based Systems IEEE Conference, SITIS 2005 (2005)
7. de Bruijn, J., Lausen, H.: Web service modeling language (WSML). Member submission, W3C (June 2005)
8. Maedche, A., Motik, B., Silva, N., Volz, R.: Mafra — a mapping framework for distributed ontologies. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, p. 235. Springer, Heidelberg (2002)
9. Omelayenko, B.: RDFT: A mapping meta-ontology for business integration. In: Proceedings of the Workshop on Knowledge Transformation forthe Semantic Web (KTSW 2002), Lyon, France, pp. 76–83 (2002)
10. Hepp, M.: GoodRelations: An ontology for describing products and services offers on the web. In: Gangemi, A., Euzenat, J. (eds.) EKAW 2008. LNCS (LNAI), vol. 5268, pp. 332–347. Springer, Heidelberg (2008)
11. Fellbaum, C., et al.: WordNet: An electronic lexical database. MIT Press, Cambridge (1998)
12. Olson, D., Delen, D.: Advanced data mining techniques. Springer, Heidelberg (2008)