



*FAST AND ADVANCED STORYBOARD TOOLS*

*FP7-ICT-2007-1-216048*

*http://fast.morfeo-project.eu*

# Deliverable D2.4.2

# Mediation amongst ontologies: Application to the FAST platform

Oszkar Ambrus, NUIG

Date: 20/02/2010

FAST is partially funded by the E.C. (grant code: FP7-ICT-2007-1-216048).

# Version History

| Rev. No. | Date | Author (Partner) | Change description |
|---|---|---|---|
| 0.1 | 07.01.2010 | Oszkar Ambrus (NUIG) | initial draft |

## Executive Summary

Ontology mediation is a critical problem in heterogeneous environments, since inter-operating services and agents often use different ontologies, and the differences between them need to be determined and overcome to allow for a seamless data exchange between these parties. This is a difficult ask, with which web-based agents must cope.

In D2.4.1 we reviewed current work and literature, discussing the main areas of the topic, that is the core problem of ontology mediation, the types of mismatches that occur between ontologies, and the three main ontology mediation approaches, namely (1) the representation of correspondences between ontologies, (2) the (semi-)automatic process of discovering these correspondences and (3) creation of a new ontology from the source ontologies.

Based on the theoretical foundations laid down in D2.4.1 we present our approach to integrating ontology matching into the FAST platform. We present the roles related to ontology matching in FAST, and the way ontology matching is achieved within each one of those roles.

# Document Summary

| | | | |
|---|---|---|---|
| **Code** | FP7-ICT-2007-1-216048 | **Acronym** | FAST |
| **Full title** | Fast and Advanced Storyboard Tools | | |
| **URL** | http://fast.morfeo-project.eu | | |
| **Project officer** | Annalisa Bogliolo | | |

| | | | | |
|---|---|---|---|---|
| **Deliverable** | **Number** | D2.4.2 | **Name** | Mediation amongst ontologies: Application to the FAST platform |
| **Work package** | **Number** | 2 | **Name** | Definition of Conceptual Model |

| | | | | |
|---|---|---|---|---|
| **Delivery data** | **Due date** | 28/02/2009 | **Submitted** | 27/02/2009 |
| **Status** | | | final | |
| **Dissemination Level** | Public ☒ / Consortium ☐ | | | |
| **Short description of contents** | D2.4.2 is the second iteration of the FAST ontology matching approach. The firsty iteration contained a general overview of the literature regarding ontology mediation, paving the way for futher analysis of the scenarios arising in FAST that require ontology mediation. Based on that general overview, this current interation presents the approach to implement ontology matching within FAST. We present the different roles within the platform, tasks and challenges related to them, and the way ontology matching is integrated within each role. | | | |
| **Authors** | Oszkar Ambrus, NUIG | | | |
| **Deliverable Owner** | Oszkar Ambrus, NUIG | | **email** | oszkar.ambrus@deri.org |
| **(Partner)** | | | **phone** | +353 91 495112 |
| **Keywords** | FAST, ontology mediation, ontology matching, ontology alignment | | | |

# Table of contents

# 1 Introduction

**Goal and Scope.** FAST 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”. In the services created within the FAST environment, the ontologies come from different parties involved in the creation of gadgets and building blocks. These ontologies, though different, will sometimes cover the same domain, making ontology matching necessary, as the means to reconcile differences in the various conceptualisations.

We aim to integrate ontology matching into the FAST platform to allow for discrepant systems to intercommunicate. We try to find a solution to these differences which range from simple naming differences to more complex data heterogeneity, behavioural differences, discrepant paradigms and other disparities. The reconciliation of these differences is called ontology matching (often mentioned in the specialised literature as ontology mediation, or ontology alignment, thus we will sometimes use these terms as well).

**Structure of the Document.** Part I describes the approach of integrating ontology matching into the FAST environment. Part II contains the first iteration of this document, D2.4.1, laying down the theoretical foundations of ontology matching. We discuss ontologies in general in Section **??**, present the ontology mediation problem in Section **??** , discuss the types of mismatches between ontologies in Section **??** and detail the main approaches to ontology mediation in Section **??**.

**Overall Approach for WPX and relation to other Work Packages and Deliverables.** Ontology matching will play an important role for the integration of different underlying resources in the user-built gadgets. We present our initial research in the implementation of ontology matching in this second iteration of D2.4, after having laid the theoretical ground work of ontology matching in the first iteration. The final version of the deliverable (i.e., D2.4.3) will finalize the implementation of ontology mediation within the FAST platform.

Part I

# Implementing Ontology Matching within the FAST Platform

# 2 Overview

## 2.1 Ontology Matching

FAST uses ontology matching 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. An ontology is used to define the vocabulary that agents use to exchange queries and assertions. If an agent commits to a common ontology, it guarantees the consistency of the queries and assertions using the defined vocabulary [Gruber, 1993]. However, different ontologies are often used in 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. The task of ontology matching is therefore critical in FAST.

Given two ontologies $O$ and $O'$ that need to be mapped to each other, we adopt the definition given in [Shvaiko and Euzenat, 2005]: 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 (=)`, `more general(`$\sqsupseteq$`)` or `disjointness(`$\perp$`)`). 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 *level 0* mapping [**?**] is a set of the above mapping elements, when the entities are discreet (defined by URIs). E.g., consider the ontology $O1$ with a class `Person`, and another ontology $O2$ with a class `Human`. For this case a matching algorithm could return the mapping element $< id_{11}, Person, Human, 0.67, => $, meaning that the `Person` class from the first ontology is found to be equivalent to the `Human` class in the second one with a confidence measure of `0.67`.

A *level 1* mapping is a slight refinement of level 0, replacing pairs of elements with pairs of sets of elements.

A *level 2* mapping can be more complex and defines correspondences in first order logic. It uses the ontology mapping language described in [**?**]. It can describe complex correspondences, such as the one detailed in Section 3.4.3 for matching the Amazon and GoodRelations ontologies.

## 2.2 Ontology Matching in FAST

The gadget life cycle in FAST has several phases and roles associated, as detailed in [**?**]. Here, we list the ones relevant for the ontology matching tasks, in decreasing order of the measure in which knowledge about ontologies is required. (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. We also mention the *resource developer*, (who, in some cases, can be the same person as the one fulfilling the ontology engineer role), who uses these ontologies to annotate the the 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 is performed in this phase, but rather the simple possibility of matching. (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, that combines the different services, but is unaware of the underlying resources and ontologies or the matching process. Thus we also need ontology matching at the gadget's run-time, where the mapping will have to be integrated into the actual JavaScript code of the running FAST gadget.

# 3 Ontology Engineering Phase

## 3.1 Alignment Tool

An ontology mapping (or alignment) is a declarative specification of the semantic overlap between two ontologies [de Bruijn and Lausen, 2005]. It is the result of the ontology alignment process. This mapping is represented as a set of axioms in a mapping language. The mapping process has three main phases: (1) discovering the mapping (alignment phase), (2) representing the mapping and (3) exploiting the mapping.

To accomplish this we need a tool to assist the ontology engineer in the ontology mapping process. Based on the description given in Sect. 2.2 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 should be supported, since we only consider these two languages. (iii) The tool should be as independent as possible, performing the alignment process with little or no user interference. This is an important requirement, since FAST is end-user oriented. (iv) It should have an open source license, allowing its free use and modification, because it needs to be integrated into the free and open FAST platform. (v) The code should be suitable for porting to other languages (in particular JavaScript), allowing it to be integrated into the FAST environment. (vi) It should be well documented.

We have considered three tools, two of which we have already detailed in [**?**].

**MAFRA** [Maedche et al., 2002] supports an interactive and incremental process of ontology mapping. It provides an explicit notion of semantic bridges. This representation is serialisable, portable and independent from the mapped languages. The bridges, however, have been designed to be used within the MAFRA system, and the alignment process needs to be done through the provided GUI.

**RDFT** [Omelayenko, 2002] is a small language originally designed to map between XML and RDF. The results are mappings represented in DAML+OIL, that can be executed in a transformation process. No hints are given to add alignment methods or extending the format and, as a major downside, we have not been able to locate the tool for download and testing.

**Alignment API** [**?**] is the tool best matching the requirements, satisfying all the desired conditions. It also being widely used at the annual ontology matching evaluation campaigns and is under active development. It 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. One of the major advantages is that it can generate the mapping result in XSLT, making it possible with the use of a simple tool, or a JavaScript package to transform XML data from one ontology into the other, independently from the alignment system.

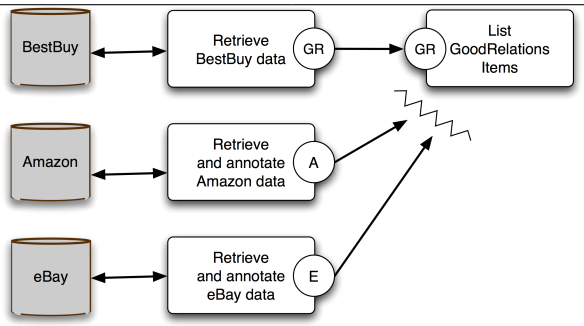


Figure 1: The scenario: three components retrieving incompatible data

## 3.2 Scenario Description

The scenario we want to test in FAST is taken from the e-commerce domain, based on the often-met case where a commercial agent aggregates data from other commercial websites. We consider the situation where a user needs a gadget which combines 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, Amazon and eBay[1], along with BestBuy, which is very close behind the former two. BestBuy is an interesting case, because it exposes its data in RDF using the GoodRelations (GR) ontology [**?**], which has recently gained a lot of popularity. It is therefore one of the first major e-commerce sites to provide semantic metadata.

Figure 1 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 and the eBay ontology. Another component displays GR items for display to the user. Out of the box, only the BestBuy retrieval and the display component can be combined. However, if the gadget designer wants to aggregate data from all three services in the display, there will have to be a mapping present between the Amazon and eBay on the one hand, and the GR ontology on the other.

[1] Based on the Alexa ranking from http://alexa.com/topsites/category/Top/Shopping

## 3.3 Ontologies

We have used three ontologies to annotate underlying resources in FAST, the first of which, namely GoodRelations, is an extensive ontology for e-commerce. The other two, i.e., the Amazon and the eBay ontologies were developed by us as simplified versions of what would be used in the real-life scenarios. Nevertheless, they are sufficient for testing the feasibility of the scenario and the options presented by current ontology alignment methods.

**GoodRelations:** This ontology is aimed at annotating so-called "offerings" on the Web, which can be products or services. The ontology is available under the Creative Commons Attribution 3.0 license and features support for ranges of units, measurements, currencies, shipping and payments, common business functions (sell, lease, repair, etc.) and international standards (ISO 4217 or UNSPSC) and codes (e.g., EAN or UPC) in the field.

```
:Offering_8794691      a gr:Offering;
  gr:hasPriceSpecification
      :UnitPriceSpecification_8794691_1;
  gr:includesObject :TypeAndQuantityNode_8794691_1;

:UnitPriceSpecification_8794691_1      a gr:UnitPriceSpecification;
  gr:hasCurrency "USD"^^xsd:string;
  gr:hasCurrencyValue "749.99"^^xsd:float;
  gr:hasUnitOfMeasurement "C62"^^xsd:string;

:TypeAndQuantityNode_8794691_1      a gr:TypeAndQuantityNode;
  gr:amountOfThisGood "1.0"^^xsd:float;
  gr:hasUnitOfMeasurement "C62"^^xsd:string;
  gr:typeOfGood :ProductOrServicesSomeInstancesPlaceholder_8794691 .

:ProductOrServicesSomeInstancesPlaceholder_8794691 a gr:ProductOrServicesSomeInstancesPlaceholder;
  gr:hasEAN_UCC-13 "0013803096095"^^xsd:string;
  gr:hasMakeAndModel :PoSM_8794691;
  rdfs:comment "With␣Auto␣Optimization␣and␣[...]"@en;

:PoSM_8794691      a gr:ProductOrServiceModel;
  gr:hasManufacturer <bbuy:Manufacturer_Canon>;
  rdfs:label "Canon␣EOS␣Digital␣Rebel␣[...]"@en;
  rdfs:comment "With␣Auto␣Optimization␣and␣[...]"@en;
```

Listing 1: Basic GoodRelations data in N3 notation

The ontology (see List. 1 for an example of GR data) is centred on the `Offering` class, which represents an announcement by a `BusinessEntity` to provide 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. This top-level concept has sub-classes representing actual product instances, product models and dummy product placeholders. A product is described by its title and description, manufacturer, make and model, etc. While GoodRelations offers terminology to allow a much higher expressivity, we will restrict the discussion to those terms relevant for our scenario.

**Amazon Ontology:** We have created a small Amazon ontology based on the datatypes supported by the web service exposed by Amazon to third-party agents to use its services. It is a simplified version of what would be needed in a real-world application, but it suffices for the scope of this work. The ontology describes Items based on the ItemAttributes description given in the Amazon Product Advertising API documentation[2].

```
: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 2: Simplified Amazon ontology data in N3 notation

The ontology features three classes for describing a product. Example instance data is given in List. 2. (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/

**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 given API. Like the Amazon ontology, it is a simplified case of what would be used in a real-world implementation.

```
:SimpleItem_1320648      a ebay:SimpleItem;
  ebay:hasItemID "E012Y090912";
  ebay:hasBidCount :"4";
  ebay:hasCountry "UK";
  ebay:hasCurrentPrice :Price_1320648_1;
  ebay:hasPrimaryCategoryName "Electronics";
  ebay:hasTitle "Canon␣Digital␣Rebel␣XSi␣[...]" .
  ebay:hasDescription "Camera␣is␣practically␣unused.␣It's␣like␣new."

:Price_1320648_1      a ebay:CurrentPrice;
  ebay:hasAmount "575.55";
  ebay:hasAmountType "GBP"
```

Listing 3: Simplified eBay ontology data in N3 notation

The ontology features three basic classes, (see Listing 3 for an example of ebay data) (1) `SimpleItem` represents an eBay `Item`, that is sold by a `SimpleUser`. It is described by a title, a `CurrentPrice` (specifying the highest bid, or the selling price of fix-priced items), 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 (which supports major international product codes — this property is from the Finding API). (2) The `CurrentPrice` features a `hasAmountType` property, specifying the currency code, and a `hasAmount` property, which is the amount of money for a price per unit. (3) `SimpleUser` contains information about eBay users. Users are 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.

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

## 3.4 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 GoodRelations ontology and the other two ontologies (see Sect. 3.4.3 for details). After studying the approach of the GR ontology, and examining the matching results between the Amazon and GR ontologies, we have concluded that automated matching between the two is unfeasible using the string-based methods employed using the Alignment API tool in this testing. Therefore, we have have created a manual mapping for this case, and have tested the automated matching on the Amazon–eBay pair, which closely resemble each other in their approach to capturing product details.

### 3.4.1 The Matcher

The matching tool called Alignment API [**?**] has been developed to support several kinds of algorithms and methods, and provide an efficient environment for testing alignment systems.

**The Algorithm:** We have used a simple string distance-based algorithm provided by the tool, which computes the string distance 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 [**?**] distance using the JWNL library with Wordnet.

**The Alignment Description:** The alignment description is given based on a simple vocabulary, containing a pair of ontologies and a set of correspondences, which express relations holding among 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. 4. Level 2 mappings

were also used for more complex, manually-created representations, as detailed in Sect. 3.4.3.

```
[ a :Cell;
  :entity1 amzn:hasCurrencyCode;
  :entity2 ebay:hasAmountType;
  :measure "1.0"^^xsd:float;
  :relation "=" ]
```

Listing 4: Level 0 mapping element example

**Using the Tool:** The Alignment API tool can be used through a GUI, as a server or from the command-line interface, of which we have chosen the last one. The tool reads two RDF/OWL ontologies, computes the alignment between them, performs some thresholding and displays the results. It can render the output in a number of formats, including HTML and XSLT.

An additional feature of the tool is its ability to evaluate results based on a reference alignment, and output the evaluation results in a table, or plot them as LaTeX graphs.

### 3.4.2 Testing Procedure

We set up the ontologies according to Sect. 3.3. For the Amazon–eBay pair we set up a reference alignment, against which the results are evaluated.

```
$ java -jar lib/procalign.jar amazon.owl ebay.owl
    -o results/equal.rdf
```

Listing 5: Executing the alignment tool on a pair of ontologies

Using the tool's command line interface, we have performed the following steps:

- generate the RDF alignment output for the four 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`.
- render the XSLT file for the SMOA distance function;

- transform an example dataset using the XSLT transformation rules;
- generate the tables and graphs displaying different parameters of the evaluation.

### 3.4.3 Results

**GoodRelations:** The GoodRelations ontology employs a unique paradigm, different from the paradigms of Amazon and eBay. In GR everything is centred around an instance of `Offering` and a graph of other instances attached to it, whereas for Amazon (and similarly for eBay), the main class is `Item`, which holds all relevant properties. In principle, `Item` would correspond to `ProductOrService` in GoodRelations, 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. 2.1) (the mapping has been created between Amazon and GoodRelations, since the eBay and Amazon ontologies are very similar in approach, and mapping eBay to GoodRelations would be an indentical task). This mapping description can later be used by the run-time JavaScript code. List. 6 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

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

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

```
<Cell rdf:about="MappingRule_01">
```

```
<entity1><omwg:Property rdf:about="&amzn;hasEAN"/></entity1>
<entity2>
 <omwg:Property>
   <first><Relation rdf:about="&gr;includesObject"/></first>
   <next><Relation rdf:about="&gr;typeOfGood"/></next>
   <next><Property rdf:about="&gr;hasEAN_UCC_13"/></next>
 </omwg:Property>
</entity2>
<measure rdf:datatype="&xsd;float">1.0</measure>
<relation>Equivalence</relation>
</Cell>
```

Listing 6: Fragment of the Amazon–GoodRelations mapping

Using this representation, complex correspondences can be modelled, using first order logic constructs. We will further investigate the possibilities of converting the level 2 mapping to XSLT, so it can automatically be used in the final JavaScript implementation. In case this is not possible, we will represent the mapping elements in SPARQL.

**Alignment 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 [**?**]. 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 column (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 (Equal) 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 (lower precision); the thresholds allow for low-scored cases to be eliminated, although this results in the exclusion of some correct

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

| Reference | | | | Equal | | | | SMOA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Prec | Rec | Fall | FMeas | Prec | Rec | Fall | FMeas | Prec | Rec | Fall | FMeas |
| 1.00 | 1.00 | 0.00 | 1.00 | 1.00 | 0.38 | 0.00 | 0.55 | 0.43 | 0.75 | 0.57 | 0.55 |

| Levenshtein | | | | JWNL | | | |
|---|---|---|---|---|---|---|---|
| Prec | Rec | Fall | FMeas | Prec | Rec | Fall | FMeas |
| 0.4 | 0.75 | 0.6 | 0.52 | 0.67 | 0.75 | 0.33 | 0.71 |

correspondences. The last column (JWNL) contains the results of the Wordnet-enabled method, which shows quite an improvement, due to the lexical analysis, which performs a much more relevant comparison of strings, giving a high number of correct results. We can clearly see how it is quite close to the reference alignment, having a correctness of 0.67 and a recall of 0.75. 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. This means that through some user assistance or an initial input alignment the tool can achieve 100% correct results.

**XSLT:** One of the appealing features of the Alignment API is that it can render the results in XSLT, thus specifying a set of rules which make it possible to transform an XML file from one format to the other. We have generated such an XML transformation description, and with the simple Linux application `xsltproc` (run from the command line) we have managed to transform data (limited, of course by the incompleteness of the alignment) from the Amazon ontology into data annotated by the eBay vocabulary.

# 4 Gadget Building Phase

Every time ontology matching is performed and an alignment is created between two underlying resources, the corresponding alignment rules are stored and an operator is created in the catalogue. At the gadget building phase, whenever two screens are combined, the catalogue will provide the possible matching operators for the two screens. This way an operator can be placed between two screens that would otherwise be incompatible.

# 5 Runtime

Employing alignment rules in the runtime phase of the gadget.

## Part II

# D2.4.1: Theoretical Foundations for Ontology Mediation

## 6 Summary

With the Semantic Web winning more and more ground both in research and industry, the mismatches between ontologies used by the inter-operating systems present an increasing problem. Ontology matching deals with identifying these mismatches, and overcoming them.

In the first iteration (described in Part II we laid the theoretical foundations of ontology matching. After presenting the main problem, and the types of mismatches between ontologies, we have discussed - based on the most prominent research results in these areas - the three main approaches to ontology mediation, that is *ontology mapping* (representing correspondences between entities), *ontology alignment* (the process of finding the correspondences between ontologies) and *ontology merging* (the process of creating a new ontology based on the source ontologies that need to be reconciled).

In this second iteration we established the approach to integrate ontology matching into FAST, and have shown the implementation details as well as test scenarios. We have presented a tool for finding the alignments between ontologies, as well as a powerful language to represent them, beyond the capabilities of the tool. We have shown how this matching is done at the different phases of the gadget life cycle and the approach taken at every level.

We plan to evaluate the feasibility of ontology matching in real-life tests and finalize it for the completed version of the FAST platform.

## References

[de Bruijn and Lausen, 2005] de Bruijn, J. and Lausen, H. (2005). Web service modeling language (WSML). Member submission, W3C.

[Gruber, 1993] Gruber, T. R. (1993). Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N. and Poli, R., editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands. Kluwer Academic Publishers.

[Maedche et al., 2002] Maedche, A., Motik, B., Silva, N., and Volz, R. (2002). Mafra — a mapping framework for distributed ontologies. In *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW-2002, Madrid, Spain*.

[Omelayenko, 2002] Omelayenko, B. (2002). RDFT: A mapping meta-ontology for business integration. In *Proceedings of the Workshop on Knowledge Transformation forthe Semantic Web (KTSW 2002), Lyon, France*, pages 76–83.

[Shvaiko and Euzenat, 2005] Shvaiko, P. and Euzenat, J. (2005). A survey of schema-based matching approaches. *Journal on Data Semantics*, 4:146–171.

# Appendix A (Lists of Tables and Figures)

## List of Tables



## List of Figures