SERVICE INTEGRATION THROUGH STRUCTURE-PRESERVING SEMANTIC MATCHING

Fiona McNeill, Paolo Besana, Juan Pane, and Fausto Giunchiglia

March 2009

Technical Report # DISI-09-015

Service Integration through Structure-preserving Semantic Matching

Fiona McNeill[1], Paolo Besana[1], Juan Pane[2], Fausto Giunchiglia[2]

[1] School of Informatics, University of Edinburgh, EH8 9LE, Scotland
f.j.mcneill@ed.ac.uk, p.besana@sms.ed.ac.uk
[2] Dept. of Information and Communication Technology, University of Trento, 38050, Povo, Trento, Italy
{pane,fausto}@dit.unitn.it

Abstract. The problem of integrating services is becoming increasingly pressing. In large, open environments such as the Semantic Web, huge numbers of services are developed by vast numbers of different users. Imposing strict semantics standards in such an environment is useless; fully predicting in advance which services one will interact with is not always possible as services may be temporarily or permanently unreachable, may be updated or may be superseded by better services. In some situations, characterised by unpredictability, such as the emergency response scenario described in this case, the best solution is to enable decisions about which services to interact with to be made on-the-fly. We propose a method of doing this using matching techniques to map the anticipated call to the input that the service is actually expecting. To be practical, this must be done during run-time. In this case, we present our *structure-preserving semantic matching* algorithm (SPSM), which performs this matching task both for perfect and approximate matches between calls. In addition, we introduce the OpenKnowledge system for service interaction which, using the SPSM algorithm, along with many other features, facilitates on-the-fly interaction between services in an arbitrarily large network without any global agreements or pre-run-time knowledge of who to interact with or how interactions will proceed. We provide a preliminary evaluation of the SPSM algorithm within the OpenKnowledge framework.

BACKGROUND

The problem of automated integration of services is key to the successful realisation of the Semantic Web, or any other system where services interact with one another. So far, this has proved difficult. Global ontologies allow different services to be expressed using the same terms, which are thus understandable to all. But there are significant difficulties with the notion of a global ontology: both the relevance of terms and appropriate categorisation of those terms is very context dependent. An ontology that included all terms that could be relevant to any situation is impossible to build, impossible to reason with and would allow no flexibility for different interpretations of situations.

However, integration of services using different ontologies is difficult. The difficulties arise at two levels: in the structure of the invocation to the service and in the values passed with the invocation. A service will expect some input parameters and will return an output. Consider for example, the web service `measurement`, whose WSDL description is shown in Figure 1. Its purpose it to provide the level of water registered by a particular sensor on a grid of sensors on a particular river-side area, which can be used during an emergency to assess the conditions. It expects as the input message the location, defined as the node identifier in the grid, and the id of the sensor, and returns in the output message the measured water level and the timestamp of the measurement.

The structure, or signature, provided by input parameters and output values must be respected by a process invoking the service. However, the invoking process may have a different signature for the caller function (parameters may have different names or

```
<wsdl>

 <xsd:element name="locationtype">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="reporterID" type="string"/>
    <xsd:element name="node" type="string"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="datetype">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="month" type="int"/>
    <xsd:element name="day" type="int"/>
    <xsd:element name="hour" type="int"/>
    <xsd:element name="minute" type="int"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

 <message name="measurementRequest">
  <part name="term" type="locationtype"/>
 </message>

 <message name="measurementResponse">
  <part name="level" type="int"/>
  <part name="date" type="datetype"/>
 </message>

 <portType name="sensor">
  <operation name="measurement">
    <input message="measurementRequest"/>
    <output message="measurementResponse"/>
  </operation>
 </portType>

</wsdl>
```

Figure 1. WSDL code for a web service returning the water level measured by a sensor in a grid of sensor for preventing flooding.

they may have a different structure). For example, a caller process could be a BPEL workflow, originally developed to invoke a service called `reading`, that does not have the concept of location, but only of the reporter and node identities, and expects the level to be named differently. The invocation needs to be adapted to the new called service.

Even after the structural adaptation has been performed, the terminology used in the parameters may be defined in different ontologies in the caller process and in the service. This may cause misunderstandings or failure: for example, the water level can be returned in meters, and the caller expected feet. Translation is required.

This case focuses on the problem of matching the signature of service invocation with that of the service when they are expressed in different ontologies. It is perfectly possible to solve this problem by manually matching the expected inputs and outputs of two services – the one that the caller expected and the one that is actually called – prior to interaction. For example, Altova MapForce[1] is a system which facilitates this manual mapping. However, performing this is time consuming and not scalable. Additionally, this presupposes that one knows in advance what calls will be necessary. This is perhaps feasible in a small, static system, but in a large, dynamic system where services may be temporary, may be updated, may suffer from occasional communication breakdown, and so on, we do not wish to limit the number of services with which it is possible to interact. A better solution in this sort of environment is to automatically integrate services on-the-fly as the need becomes apparent.

Using our matching approach we are able to map between the invocation of the service, written in the ontology of the caller, and the call the service is expecting, written in the ontology of the service. The goal of the matching is two-fold:

- to establish whether these services (the expected and the called) are *similar enough*: if the service is being asked to perform an action that is too different to the function it is equipped to perform, then the correct response is to refuse to interact;
- if the call is judged to be similar enough, then an adaptor is generated to bridge beween the caller and the invoked service .

Our technique is designed to work at run-time, without user interaction and without any pre-alignment of ontologies, as we believe that in a Semantic Web kind of environment, such an approach is vital. This is therefore a lightweight and flexible approach that can be employed on-the-fly if – and only if – the need arises.

SETTING THE STAGE

Structure-preserving Semantic Matching

We have developed the *Structure-preserving Semantic Matching (SPSM)* technique, which allows us to find a map between two service descriptions and returns a score in [0 1] indicating their similarity.

The SPSM maps trees structures; we therefore first need to transform web services into trees. The name of the service becomes the root of the tree, while the parts in input and

[1] http://www.altova.com/products/mapforce/data_mapping.html

output messages become the children. As the WSDL description in Figure 1 shows, parts can contain complex structures (such as `Location` in the input message and `Date` in the output): the part itself becomes a subtree. For compactness, we will represent trees as formulae. The WSDL in Figure 1 can therefore be represented as:

```
measurement(location(ReporterID, Node), Level,
        date(Month, Day, Hour, Minute))
```

Note that in such formulae and in diagrams such as Figure 2, the names of the variables indicate the types expected. For example, the second level argument `Level` indicates that the argument is a variable that should be instantiated with a value of type `level`.

SPSM is a two-step process. Firstly, we make use of adapted conventional ontology matching techniques to investigate relationships between the individual words in the nodes of the trees. The second – novel – step matches the structure of the trees to discover an overall relationship. This is crucial because the structure of the tree itself contains a lot of semantic information that must be considered if we are determining whether two service calls are equivalent or similar. SPSM, therefore, needs to preserve a set of structural properties (e.g., vertical ordering of nodes) to establish whether two trees are globally similar and, if so, how similar they are and in what way. These characteristics of matching are required in web service integration applications, see, e.g., (Kluch et al, 2006; Li and Horrocks, 2006; Gooneratne and Tavi, 2008).

Moreover, SPSM allows us to detect not only perfect matches – which are unlikely to occur in an unconstrained domain – but also *good enough* matches. SPSM returns both a mapping between two trees and a numerical score in [0 1] indicating the degree of *global similarity* between them. A match between two trees is considered to be *good enough* if this degree of global similarity exceeds some threshold value. Since the concept of *good enough* is very context dependent – in safety critical situation perhaps only a near-perfect match will do but in other situations a much weaker match may suffice – this threshold is set by the user according to the particular interaction (Giunchiglia et al, 2008b). This approach greatly increases the range of services it is possible to interact with. This solution is lightweight enough to be done quickly on-the-fly, during run-time, so that we need have no expectations of which services we will want to interact with in advance of run-time.

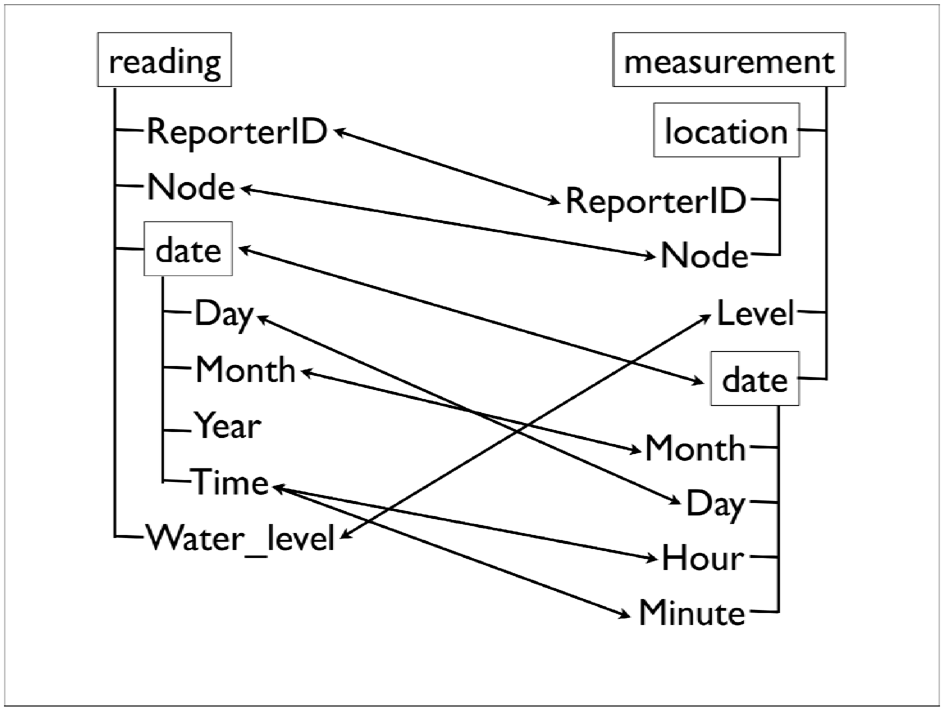

Figure 2.Two approximately matched web services as trees – *T*₁: `reading(ReporterID,Node,date(Time,Day,Month,Year),Water_level)` and *T*₂: `measurement(Level,location(ReporterID,Node), date(Month,Day,Hour,Minute).`
Functions are in rectangles with rounded corners; they are connected to their arguments by dashed lines. Node correspondences are indicated by arrows.

Service descriptions that are not written in WSDL will need to have the conversion step to turn them into trees built for them, but we believe that it is possible to view most service descriptions as tree structures and that this conversion process will generally be straightforward. An example of two service descriptions, which have been converted into trees, being approximately mapped, can be seen in Figure 2.

Once the conversion to tree has taken place, the SPSM algorithm consists of two stages:

Node Matching – this matches the nodes in one tree to the nodes of another tree. This will often be matching single words to one another, although nodes may be more complex (for example, they can be composed of more than one word) and our techniques are able to deal with this. These terms may be annotated with references to ontologies so that it is easier to determine their semantic meaning and, if so, our matching techniques take advantage of this. If there is no additional information then our matching techniques rely on syntactic properties of the terms (for example, suffixes and prefixes) and standard ontologies such as WordNet

(Fellbaum, 1998). This step is performed using the S-Match system (Giunchiglia et al, 2007). For example, in matching a tree `reading(date(Day,Month,Year),water(Level))` to a tree `measurement(Liquid,Level,date(Month,Day))`, this step would discover that the words `Date`, `Day`, `Month`, `Level` and `Reading` in the first term all have equivalent words in the second term (in the case of the first three, these are identical, whereas `Reading` is matched to the semantically equivalent `Measurement`).

Tree Matching – once we have the correspondences between the nodes of the trees, the next step is to match the whole trees and determine the global similarity between them. This is achieved by considering the relationships, within the trees, of nodes identified as similar in the previous step. For example, if we were matching `reading(Water,Level)` and `reading(Liquid,Level)`, we would expect the similarity score to be high, but the relationship between `reading(water(Level))` and `reading(Liquid,Level)` to be much lower: the different structure of these trees indicates a difference in meaning. For this step, we make use of a tree-edit distance algorithm. Tree-edit distance algorithms are designed to determine the cost of translating one tree into another through the application of three operations: *(i)* vertex deletion, *(ii)* vertex insertion, and *(iii)* vertex replacement (Tai, 1979). However, tree-edit distance algorithms do not consider the semantics behind these operations: for example, according to the standard algorithm, replacing a vertex `Water` with a vertex `Liquid` would cost the same as replacing `Water` with `Sandwich`, although it is clear that a far greater change is occurring in the second case.

We have therefore augmented a standard tree-edit distance algorithm so that the cost of performing each operation is dependent on the expense of performing the change: that is, a smaller semantic change costs less than a large change. To do this, we make use of Giunchiglia and Walsh's theory of abstraction (Giunchiglia and Walsh, 1989; Giunchiglia and Walsh, 1992), which provides a formal theory of how two first-order terms may be related and yet non-identical. For example, the number of arguments of predicates may be different; the types of these arguments may be more or less general; the name of the predicate may be more or less general. Since it is trivial to convert first-order terms into trees, this theory is applicable to our tree matching step.

Thus the node matching step tells us in what way the node terms are related and the combination of the tree-edit distance algorithm and the abstraction operations tell us how similar two trees are by combining the steps that are necessary to convert one tree to another (which is functionally identical to providing a map between the two) with the costs that should be assigned to each of these steps. Taking one example from Figure 2, the node matching tells us that `Water_level` from Tree 1 is a more specific type of `Level` from Tree 2 (or `Water_level` is a subclass of `Level`), and the augmented tree-edit distance algorithm will map these two together (as part of the process of mapping the whole tree) and, using the abstraction operations, determine that the cost of this map should be low due to the relationship identified in the node matching step. Therefore, by combining semantic matching with structural matching, we obtain the SPSM (*structure-preserving semantic matching*) algorithm.

Of all the potential maps between two trees, the tree-edit distance algorithm will return the map with the least overall cost (calculated through the application of abstraction operations). The cost of the overall map is calculated by

$$Cost = min \sum_{i \in S} k_i * Cost_i \quad (1)$$

where S stands for the set of the allowed tree edit operations; k_i stands for the number of *i-th* operations necessary to convert one tree into the other and $Cost_i$ defines the cost of the *i-th* operation. Our goal here is to define the $Cost_i$ in a way that models the semantic distance. We can then define the similarity between two trees T_1 and T_2 to be:

$$TreeSim = 1 - \frac{Cost}{\max(T_1, T_2)} \quad (2)$$

This case is intended to outline the ideas and motivation behind our ideas; for full technical details of this process, together with implementation information, see (Giunchiglia et al, 2008a).

The OpenKnowledge Framework

Matching service descriptions is only one aspect of service integration: another important aspect is service selection: how a potentially suitable service is located, how a particular one is chosen from potentially many, and so on. In this section, we introduce the OpenKnowledge framework within which the SPSM algorithm was originally designed, in order to describe how this framework allows the full process of service integration to take place, and to show SPSM in action within a specific context. Note that although SPSM was designed within this context, it is nevertheless very widely applicable: in fact, it does not need to be restricted to matching service descriptions but can be used for matching any two artifacts that can be expressed as a tree.

The OpenKnowledge framework facilitates interactions between disparate peers or services, which generally do not share an ontology or have prior knowledge of one another. The key technology that makes this possible is the use of shared choreographies called *Interaction Models (IMs).* These IMs can be designed by any user on the network, and can then be shared across the network, so that determining an IM for a particular interaction is usually a case of finding the appropriate one for reuse rather than writing one from scratch. Note that in the OpenKnowledge context, services are proactive, signing up to IMs in which they wish to play a role. Calls to services therefore do not come out of the blue, from an unknown caller, but occur when that IM is enacted and are of the form described within the IM.

The OpenKnowledge framework enables this through providing an API that can be used by an application to become a peer in a network. The API exploits:

- a *distributed discovery service*, which searches for suitable IMs for a particular interaction;
- a *matching service*, which uses the SPSM algorithm to map between requirements in IMs and abilities in the peers to determine how similar they are;
- a *trust component* to allow users to assess with which peers they wish to interact with;

We will not address these components, other than the matching service, in any detail in this case. Further information about OpenKnowledge can be found on the project webpage[2].

Describing Interactions

An Interaction Model (IM) specifies the interaction between different peer in tasks that require their coordinated activities. Interaction Models are written in LCC (Robertson, 2004), a compact, executable language based on process calculus. An IM is composed by a set of *role definitions*: a peer enters an interaction by taking a role, and follows the unfolding of the interaction as specified by the role definition. The definition prescribes to a peer in a specific role what messages to send, which messages to expect and what other roles to adopt later if necessary. The coordination of the peers is obtained through message exchange between the roles they have adopted, while the behaviour of peers is defined by constraints on messages. Through constraints it is possible to set preconditions for sending a message and for changing role as well as describing the effects of receiving a message. A peer must solve the constraints in order to proceed. The IM makes no assumptions at to how constraints are solved and the operation is delegated to the peer. In LCC constraints are expressed as first order predicates, which can be easily transformed into trees for matching.

```
a(querier,Q)::
  request(RepId,Nd) => a(sensor,S) ← needLocation(RepID,Nd)
  then
  level(Lvl,Date) <= a(sensor,S)

a(sensor,S)::
  request(RepID,Nd) <= a(querier,Q)
  then
  level(Lvl,Date) => a(querier,Q) ← reading(RepID,Nd,Lvl,Date)
```

Figure 3. A simple Interaction Model for querying a sensor about the water level.

Figure 3 shows a simple IM for querying a sensor. The IM is performed by two peers, one taking the `querier` role and the other taking the `sensor` role. The `querier` needs first to satisfy the constraint `needLocation(RepID,Nd)` to select the interested reporter ID and the node, then send the request message to the `sensor` and wait for the reply. The

[2] www.openk.org

`sensor` receives the request, satisfies the constraint `reading(RepID,Nd,Lvl,Date)` and sends back the reply. The IM execution is then concluded. Note that in an IM, there are no semantics in the message: the name of the message is merely a placeholder and the meaning of the arguments is determined within the constraints. Thus the ability to play a role depends on the ability to satisfy the constraints on the messages in that role. Any peer using the OK infrastructure can trivially pass any message if the constraints on that message are satisfied.

A constraint in an IM can be compared to the call to a web service in a BPEL workflow. In order to solve a constraint a peer needs to map it to its own local knowledge base, provided by an extensible set of plug-in components (Besana et al, 2007). The plug-in components expose methods, which can be simple wrappers for web services, or can be self-contained java methods. We have developed a tool that generates a wrapper component from a WSDL file: each operation in it becomes a method in the component.

In the constraint `reading(RepID,Nd,Lvl,Date)`, the variables `RepID` and `Nd` are already instantiated (they are received with the `request` message), and are the input parameters; the variables `Lvl` and `Date` are instantiated by the peer when solves the constraint and are the output parameters.

```
@annotation(@role(sensor), @variable(RepID, reportedID)
@annotation(@role(sensor), @variable(Nd, node)
@annotation(@role(sensor), @variable(Lvl, water_level)
@annotation(@role(sensor), @variable(Date,
date(day,month,year,time))
```

Figure 4. The annotations of the parameters used by the role `sensor` in the IM of Figure 3

While first order predicates are usually untyped, in OpenKnowledge, arguments can be annotated with their ontological type and, if desired, these types can be annotated with a reference to an ontology in which the semantics of that type are given. The annotations are used to create the trees that are then matched by SPSM. Figures 4 and 5 shows annotations for the parameters in the IM and for the method in the peer's component.

```
@MethodSemantic(language="tag",
args={"location(reporterID,node)",
      "level",
      "date(month,day, hour, minute)"}
public boolean measurement(Argument Lc,Argument Lv,Argument D){...}
```

Figure 5. The annotations for the method `measurement` in the plug-in component of a sensor peer

Lifecycle of Interaction in OpenKnowledge

The IMs are published by the authors on the *distributed discovery service (DDS)* (Kotoulas and Siebes, 2007) with a keyword-based description. Peers search and subscribe to roles in IMs in the DDS. The OpenKnowledge kernel provides the functionality needed to subscribe to a role and the framework for handling the plug-in components used to satisfy constraints. The peer can be a GUI-based application whose components interact directly with a user or a server application that solves the constraints automatically, possibly calling the web services wrapped by the components or accessing a database.

The lifecycle of an interaction is:

Interaction selection: a peer searches, by sending a keyword-based query to the DDS, for published IMs for the task it intends to perform. The DDS replies with a list of IMs satisfying the query. The peer needs to compare the received IMs with its plug-in components, in order to select the one that best matches its capabilities. This is one instance – the most important one – where the SPSM algorithm comes into play. In order for a peer to decide whether it wishes to play a role, it needs to map every constraint on that role to one of the methods in its plug-in components. For each of these constraints, the SPSM algorithm will return a numerical score in [0 1] describing how close this constraint is to one of the peer's constraint, as well as a map detailing how this conversion must be done. To estimate how good the peer will be at performing that role, it must somehow aggregate these scores. The simplest way to do this – and the way that is current implemented – is to average all scores over the number of constraints to be mapped. However, more sophisticated mechanisms could be devised which could incorporate user preferences and context-dependent information. Once this overall score has been calculated, the peer must decide whether or not to subscribe to the role. This is entirely up to the peer and it may subscribe even if it gets a very low score. In such a case, it would not usually be in the peer's interests to subscribe, as it is very likely to fail in the execution of the role. If it finds a role in an IM with an acceptably high matching score, it subscribes on the DDS, indicating its intention to perform the appropriate role in it. As part of the subscription process, it must declare a matching score.

Peers may subscribe to as many IMs as they wish, to play as many different roles as they wish. For example, in a vending scenario, a `seller` peer may subscribe to many different IMs in the `seller` role, as it may be content to act as a seller simultaneously in many different types of purchase interactions. A `buyer` would more typically only wish to buy once (though of course this depends on the exact situation), so would only wish to subscribe once in the role `buyer`, but may also be subscribed in other roles in different IMs for quite different goals. A peer may also be subscribed as seller in one purchase IM, and as buyer in another, as it may be interested in buying supplies for its production as well as in selling it.

Bootstrap: when all the roles in an IM are subscribed to, the discovery service randomly selects a peer in the network, asking it to play the coordinator of the interaction. If it accepts, it becomes the IM coordinator and asks all the subscribed peers to select which other peers they are prepared to interact with. This matching score provided by peers as they subscribe is also useful to other peers deciding whether or not they wish to interact with that peer in that role. However, neither the system nor other peers have any way of checking this matching score: the peer's own capabilities and ontology are private. So peers must use this score with caution, for there are several reasons why it may not be accurate: the peer may be dishonest and may be trying to exaggerate its abilities; it may have a poor ontology, so the matching score returned may be a poor reflection of its actual ability to perform the role. This score is therefore most useful to others when it is moderated by some kind of trust score examining the peer's past behaviour: if the peer is dishonest or inept, it will repeatedly underperform and therefore trust scores are lowered. We therefore have developed a *good enough* algorithm, whose role is to moderate the matching score with respect to a trust score. OpenKnowledge provides a built-in mechanism for calculating trust, based on prior experience of interaction in the same, similar or non-similar contexts, and a way to combine this trust score and the matching score to obtain a single score reflecting how well that peer is likely to behave. This process is explained in (Giunchiglia et al, 2008b). Peers are free to use this built-in method or to use their own mechanisms as they please.

After receiving the peers' preferences, the IM coordinator creates a group of peers who are all willing to interact with one another in their proposed roles. If the group covers all the roles, it starts the interaction. If there is more than one way of filling roles such that all involved peers are satisfied, the choice of allocation is made arbitrarily. It is thus possible that peers subscribed for roles will not be chosen in a particular run of that interaction. In such a case, they must wait for a subsequent run of the interaction, perhaps weakening their choice criteria next time, as they may be ruling themselves out of potential allocations by refusing to interact with many of the other subscribed peers.

Run of the interaction: the IM coordinator runs the IM locally: messages are exchanged between proxies of the peers, which are contacted in order to solve the constraints.

Follow-up: after the run of the interaction, the IM coordinator sends the log of the interaction to all involved peer so that they can analyse it if they wish to. The analysis can be aimed at computing a trust value for the other peers (Giunchiglia et al, 2008b) to be used in selecting peers in future interactions or to create a statistical model for the content of the messages, in order to improve mapping (Besana and Robertson, 2007). If, interaction after interaction, a peer is consistently unreliable it will be selected less and less frequently by the other peers.

In a more orchestration-oriented model, the invocations to services are normally grounded at design time by the designer of the workflow. In this model, the peers decide to take part in interactions: they can look up an interaction for a specific task, they can be alerted when new interactions are published, or they can be asked to evaluate an interaction upon the request of another peer, but in all cases they evaluate the IMs they receive and then select those they want to subscribe to. The task of handling heterogeneity is therefore distributed among the peers.

CASE DESCRIPTION – Flooding in the Trentino Region

The OpenKnowledge system has been fully evaluated in two testbeds: Proteomics and emergency response. Here, we explain the emergency response testbed and explain the role that the SPSM algorithm took in providing the necessary functionality.

Emergency response was chosen as being a particularly knowledge-intensive and dynamic application domain, with many players and a high potential for unexpected developments. We briefly outline the general scenario and then describe a specific interaction in more detail, highlighting where the techniques discussed in this paper will be utilised.

The general scenario we are exploring is the case of the flooding of the river Adige in the Trentino region of Italy, which presents a significant threat to the city of Trento and the surrounding area and which has occurred seriously many times before, most notably on November 4th, 1966. We have large amounts of data from the 1966 flood, as well as data concerning the emergency flooding response plans of the Trentino authorities. Around this data, we have developed scenarios of interacting peers: for example, coordination centres, emergency monitoring services, the fire brigade, sensor nodes, GIS systems, route finding services and weather services.

Emergency response is not inherently peer-to-peer: we would of course expect that the key players would have strategies worked out well in advance and would have established the infrastructure and vocabulary for communicating with other key players. However, the chaotic nature of an emergency means that many players who will not have been able to coordinate in advance, or who were not expected to participate, may become involved. Additionally, services which were part of an emergency response may be unexpectedly unavailable or may be swamped by requests, and in such a situation, it is crucial that the emergency response can carry on regardless. Additionally, services may develop and change and it is unrealistic to expect these changes would always be known and accounted for in advance.

The e-Response system we have developed for this testbed is used:

i) to model and execute interactions between peers involved in an emergency response activity, whether individuals, sensors, web services or others;
ii) to provide feedback about the environment at appropriate moments, in a way that mirrors the real world (for example, a peer attempting to take a road will be informed that the road is blocked only when it is actually at that road, and it can then share this information with other peers through the network).
iii) to visualize and analyze a simulated coordination task through a Graphical User Interface (GUI).

The developed e-Response system is composed of two major components: the e-Response simulator and the peer network (and related interaction models). The e-

Response simulator provides the disaster scene and its evolution, thus representing the “real world” within which all the actors (network peers) acts. The idea is that once the simulator has been used to aid the development and thorough testing of the approach, it could be removed and the peer network could instead operate in a real world situation (using real peers rather than simulated ones).

Every peer (either simulator or network peer) has an OpenKnowledge plug-in component (the OpenKnowledge kernel) which enables it to publish and search for IMs and be involved in a coordination task with other participants. Some of the peers in the peer network interact with both the simulator and network peers: these are the peers that ‘exist’ in the physical location. These peers will usually receive sensory information and be able to directly influence the simulated world, though some can only do one or the other (for example, water-level sensors have some sensory ability (they will receive information about the water-level) but they cannot directly influence the world, they can only indirectly influence the world by sharing this information across the peer network). Other network peers communicate among themselves and never connect to the simulator: these are peers that are not physically involved in the simulation and cannot directly affect the world, such as geographical-map-provider peers.

The peer network reconstructs (in a limited form) the infrastructure of the emergency response: for example, the command centre that will control the whole response (except in unforeseen circumstances such as it becoming uncountable), the fire teams they will be commanding, the buses that are to evacuate the citizens, and so on. One important job of the command centre is to keep a picture of the changing environment that is as accurate as possible and this is done through gathering information from other peers.

Figure 6 illustrates a scenario in which some of the peers in the simulation are interacting in order to facilitate the evacuation of citizens by bus from the flooding area. The emergency coordinator (EC) communicates with the buses (Bs), informing them which area they should pick up from and which area they should evacuate the citizens to. It is up to the buses to determine an appropriate route for this. To do this, they can communicate with the route service (RS), which will tell them what possible routes there are, but since they are in a flooding situation, they must also try to establish which routes are closed. To do this, they communicate with the civil protection peer (CP), whose role is to continually poll the water-level sensors (r_1, r_2, …) as to the water level in their vicinity, and from this they can calculate whether a route suggested by a bus is accessible. As long as this process is functioning, it is perhaps reasonable to assume that there will be no difficulty with integration: this is part of the planned emergency response, and such service integration should have been calculated in advance (though even this much cannot be certain as the peers may be constantly evolving – perhaps the owners of the water-level sensors have upgraded their ontologies since the most recent coordination effort). However, in such an emergency situation, such structure is not necessarily reliable. Perhaps the civil protection peer will be swamped with requests from buses, citizens and others and be unable to respond to some of them; perhaps it will crash; perhaps it is housed in a building that is itself being flooded. If a bus peer cannot reach the civil protection peer, they must still do their best to reach their given destination. The OpenKnowledge system allows the seamless change from a centralised system to a decentralised one: instead of interacting with the civil protection peer, a bus can communicate with the water-level sensors directly and calculate for themselves which routes are possible. However, it is highly unlikely that such a scenario has been planned for in advance, and we are therefore forced to perform service integration on-the-fly.

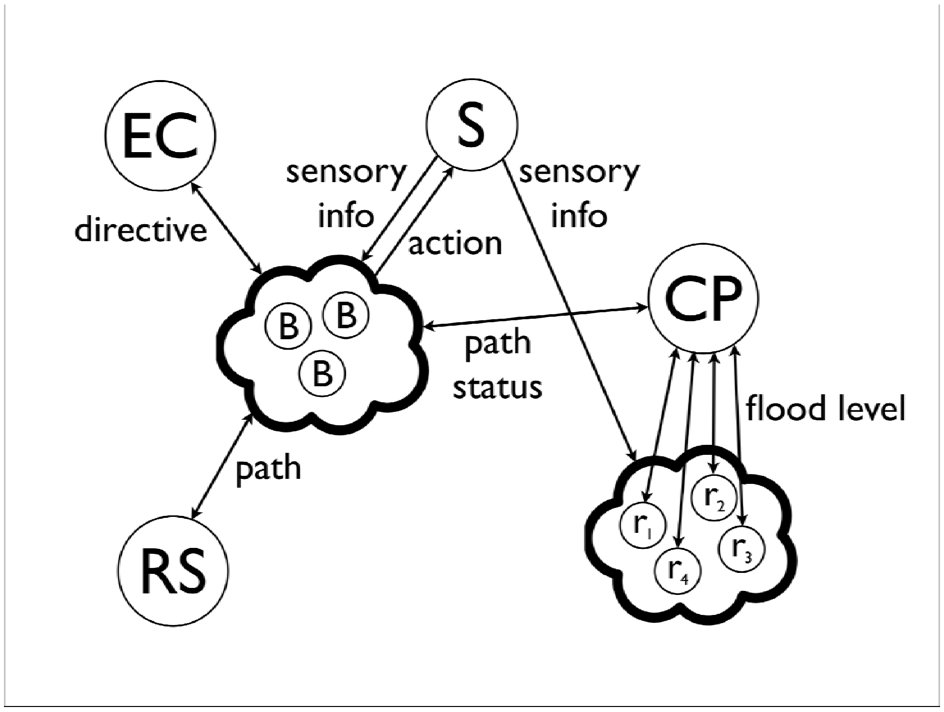

Figure 6. Interactions of peers in evacuation scenario.

We repeat the lifecycle of interaction described above for this particular interaction.

Interaction selection: both the bus peer and the water-level sensor peers must be subscribed to an appropriate IM to play their relevant roles. Since the bus peer is taking the initiative here, this is most likely to happen through the bus peer searching for and subscribing to an IM in which many water-level sensor peers are already subscribed. A well-organised water-level sensor should subscribe to many such IMs so that it is ready to perform its role whenever requested. There may be many sensors at a single node, so in order to determine the water level at that node, it is necessary to choose one of potentially many with which to interact. The IM used in such a situation could be the one described in Figure 3, and let's imagine the process by which these water-level sensors have subscribed to their role. Since they wish to play the role sensor, they have only one constraint to satisfy: `reading(RepID,Node,Level,Date)`. Imagine six sensors wish to sign up, and they describe their abilities in the following ways:

i) `measurement(RepID,Node,Level,Date)`

ii) `reading(ReporterID,Level,Node,Date)`
iii) `reading(RepID,Level,UnitMeasure,Node,Date)`
iv) `reading(RepID,Water_level,Node,Date)`
v) `output(Level,Node)`
vi) `measurement(location(ReporterID,Node),Level,Date)`

They would each use the SPSM algorithm to map their abilities to the constraint. Sensor ***i)*** would discover that had a perfect match; all that is necessary is to consider measurement to be equivalent to reading and the node matching step reveals this is permissible. Sensor ***ii)*** would have less than perfect matching because it has to infer a match between `RepID` and `ReporterID`. Analysis of these terms would indicate a high similarity but it is not certain, in the absence of further information, that they are intended to refer to the same thing. The adapter returned would also switch the two central arguments, matching `Level` to `Level` and `Node` to `Node`. This mapping would not influence the similarity score, as the order of nodes is assumed not to be semantically significant. Sensor ***iii)*** would have a high but less than perfect matching score as it has an extra argument that would not be used: `UnitMeasure`, which is intended to make explicit the units in which the measurement is given. Sensor ***iv)*** would also have a lower matching score, losing points through the match between `Water_level` and `Level`. In this particular situation, we can see that these are functionally equivalent because what is meant by `Level` is the level of the water. However, the water-level sensor does not have this high-level view and therefore cannot be sure of this. Sensor ***v)*** would receive a very low score: the naming of its predicate is unintuitive and does not describe what it actually does, and it misses out key information such as its ID and the Date. Sensor ***vi)*** is a slightly simplified version of the one discussed earlier in the case and illustrated in Figure 2. There is a significant structural difference here, also illustrated in Figure 2, in that an extra predicate `location` is included, and the arguments `ReporterId` and `Node` become children of that predicate, and grandchildren of the top-level predicate `measurement`. The similarity score would be lower due to this: in fact, this is an 'organisational' detail and does not really affect the meaning of the arguments, but it is an indication that this meaning may be different. Nevertheless, every argument in the constraint can find a similar or exact argument to match to in ability, albeit in a different structure. Therefore the similarity score would be reasonable but not as high as for sensors ***i) - iv)***.

These sensors will all subscribe to play the role `sensor` and the bus peer will subscribe to play the role `querier`.

Bootstrap: If these sensors are all for at the same node, then, when the coordinator asks the bus peer which peer it wishes to interact with, it need only chose one. Unless it has any information to the contrary, and assuming the sensor peers are all honest, it will probably choose sensor ***i)***, as it has the highest matching score. However, if it has previously interacted with sensor ***i)*** and found it to be unreliable, it will have a low trust score, and so the GEA score, formed by combining its matching and trust scores, may be lower than the GEA scores of other sensors. The bus peer would normally choose the peer with the highest GEA score to interact with. Assuming the chosen sensor is happy to interact with the bus peer, the interaction will proceed.

Run of the interaction: The appropriate messages are passed.

Follow-up: The bus peer will update its trust model according to the outcome of the interaction. If the interaction was successful, the bus peer's trust in the sensor peer will increase. If it fails, it will be lowered. For example, sensor ***v)*** may claim to have a very high matching score and may come out top in the GEA calculation. However, it will fail to satisfy the constraint on its role and so will not be able to pass the message, leading to failure. The next time this interaction occurs, it is much less likely to be chosen even if it reports a high matching score, as the trust score will be low. Alternatively, the interaction may run smoothly – if, for example, sensor ***i)*** was chosen – but if sensor ***i)*** is faulty it will report a false level, leading to a successful interaction but an unsatisfactory outcome. This will not be as obvious as a breakdown of the interaction, but once it has been noticed by the bus peer (perhaps when it finds its route unexpectedly flooded), the bus peer will update its trust model accordingly. The bus peer is also free to share its trust model with other peers if desired.

Evaluation

There are many reasons why heterogeneity can become a problem, even amongst services that were originally designed to interact. For example, drifting can cause heterogeneity between components and IMs: components that were designed for a particular interaction could be used in other interactions and can change over time to adapt to these, when they are reused in the original interaction, matching is required. Similarly, interactions designed for a specific context may be used for different aims and therefore adapted to better suit these aims. Moreover, new interactions or components can be developed by copying others.

Starting from these assumptions, we tried to evaluate how the matching mechanism, described previously, could cope with these sort of heterogeneity. The evaluation aimed at exploring the robustness of the SPSM approach towards both typical syntactic alterations (i.e. replacements of node names, modification of node names and misspellings) and typical meaning alterations (i.e. usage of related synonyms, hyponyms, hypernyms) of node names.

Since the tree alterations made are known, these provided the reference results. This allows for the computation of the matching quality measures, such as Precision (which is a correctness measure) and Recall (which is a completeness measure). The alterations are applied probabilistically on each node of the original tree: increasing the probabilities of the modifications it is possible to obtain trees that are statistically more and more distant from the original one. The tree alteration procedure has been inspired by the work in (Euzenat and Shvaiko, 2007) on systematic benchmarks.

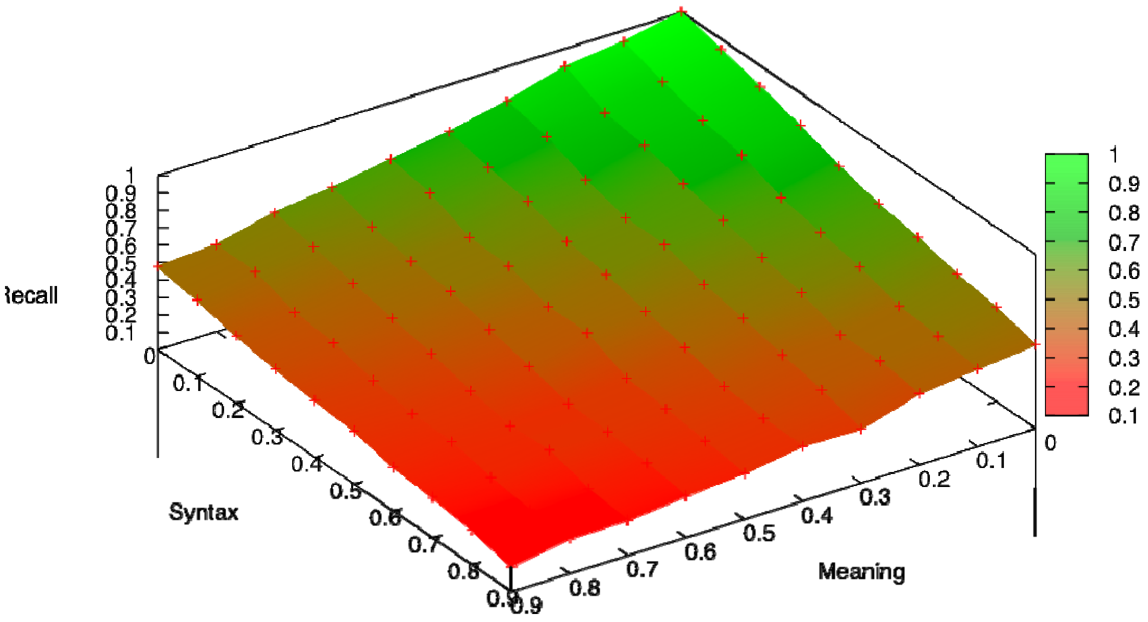

Figure 7. Recall results for syntactic and meaning alterations

Figure 7 shows how recall behaves when the probabilities of syntactic and semantic alterations are increased. Recall decreases slowly: only when both semantic and syntactic changes are extremely likely, recall drops to 0.1. In our experiments, precision was always very high. This is not uncommon in matching scenarios, where recall is often the problem.

The evaluation is done by comparing the output of the SPSM algorithm with the output of a standard tree-edit distance algorithm, which does not consider the semantics. The fact that SPSM performs better, whilst a reassuring validation of the approach, is therefore not particularly surprising. A more powerful approach would be to compare our work against a 'state-of-the-art' system. However, as far as we believe there is no other approach currently existing that can be used to perform the same task as SPSM: matching trees whilst considering the semantics. Systems that perform semantic-free tree matching can be compared to SPSM in such experiments, as they are at least capable of performing the necessary tree matching. Other semantic-based approaches cannot do this, and therefore will fail completely in the task.

We are currently undertaking a much more thorough evaluation of the whole process of service integration within OpenKnowledge. These results may lend themselves more naturally to comparison with other service integration approaches because the scope will not be so limited as in the current experiments. We intend to publish these results shortly.

RELATED WORK

Our work builds on standard work in tree-edit distance measures, for example, as espoused by (Shasha and Zhang, 1997). The key difference with our work is the integration of the semantics that we gain through the application of the abstraction and refinement rules. This allows us to consider questions such as *what is the effect to the overall meaning of the term (tree) if node a is relabelled to node b?,* or *how significant is the removal of a node to the overall semantics of the term?* These questions are crucial in determining an intuitive and meaningful similarity score between two terms, and are very context dependent. Altering the costs assigned to the tree-edit distance operations enables us to provide different answers to these questions depending on the context, and we are working on giving providing even more subtle variations of answers reflecting different contexts.

Work based on these ideas, such as Mikhaiel and Stroudi's work on HTML differencing (Gligorov et al, 2005), tends to focus only on the structure and not on the semantics. This work never considers what the individual nodes in their HTML trees mean and only considers context in the sense that, for example, the cost of deleting a node with a large subtree is higher than the cost of deleting a leaf node; the semantic meanings of these nodes is not considered.

Many diverse solutions to the ontology matching problem have been proposed so far. See (Shvaiko and Euzenat, 2005) for a comprehensive survey and (Euzenat and Valtchev, 2004; Euzenat and Shvaiko, 2007; Noy and Musen, 2003; Ehrig et al, 2005; Gligorov et al, 2007; Bergamaschi et al, 1999; Kalfoglou and Schorlemmer, 2003; Straccia and Troncy, 2005) for individual solutions. However most efforts has been devoted to computation of the correspondences holding among the classes of description logic ontologies. Recently, several approaches allowed computation of correspondences holding among the object properties (or binary predicates) (Tang et al, 2006). The approach taken in (Hu and Qu, 2006) facilitates the finding of correspondences holding among parts of description logic ontologies or subgraphs extracted from the ontology graphs. In contrast to these approaches, we allow the computation of correspondences holding among trees.

The problem of location of web services on the basis of the capabilities that they provide (often referred as the matchmaking problem) has recently received considerable attention. Most of the approaches to the matchmaking problem so far employed a single ontology approach (i.e., the web services are assumed to be described by the concepts taken from the shared ontology): see (Klusch et al, 2006) for example. Probably the most similar to ours is the approach taken in METEOR-S (Aggarwal, 2004) and in (Oundhakar, 2005), where the services are assumed to be annotated with the concepts taken from various ontologies. Then the matchmaking problem is solved by the application of the matching algorithm. The algorithm combines the results of atomic matchers that roughly correspond to the element level matchers exploited as part of our algorithm. In contrast to this work, we exploit a more sophisticated matching technique that allows us to utilise the structure provided by the first order term.

Web services composition follows two alternative approaches: *orchestration* or *choreography*. Their primary difference is their scope. An orchestration model provides a scope specifically focussing on the view of one participant. Instead, a choreography model covers all parties and their associated interactions giving a global view of the system. The OpenKnowledge system is closer to the choreography approach, since all services involved know – and can choose – with whom they are interacting and what these interactions will involve (once they have signed up to IMs; prior to run-time this may not be known). Other important service composition languages are BPEL[3] and YAWL (van der Aalst and ter Hofstede, 2005) (orchestration languages) and WS-CDL[4] (a choreography language). BPEL and YAWL benefit from the simplicity of the orchestration approach, but the OpenKnowledge system has advantages: services choose to take part in interactions and they know in advance both what these interactions will involve and which other services they may be interacting with, allowing them to make informed decisions as to whether this is in their interests and which other services they would prefer to participate with. Additionally, the interactions are not owned by any particular service and are therefore not biased towards any one service but rather allow free interaction for all. Crucially, this approach is also scalable, allowing a network of arbitrarily large size to interact on the OpenKnowledge system. WS-CDL is closer to the OpenKnowledge approach but, unlike OpenKnowledge, it is merely a specification and is not executable.

In summary, much work has been done on structure-preserving matching and much has been done on semantic matching, and our work depends heavily on the work of others in these fields. The novelty of our work is in the combination of these two approaches to produce a structure-preserving semantic matching algorithm, thus allowing us to determine fully how structured terms, such as web service calls, are related to one another.

CURRENT CHALLENGES

The current implementation of the SPSM algorithm, though it has proved effective in practice, does not have the full scope we believe to be necessary. For example, it assumes that matching between the abilities of a peer and the requirements of a role can be performed by considering a one-to-one relationship between arguments. If, for example, we were to match:

```
reading(RepID,Node,Date,Level)
```

[3] Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[4] Web Services Choreography Description Language Version 1.0, http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/

to
`reading(RepID,Node,Day,Month,Year,Level)`
then, once `RepID, Node` and `Level` had been matched, a choice would need to be made as to whether to map `Date` to `Day`, `Month` or `Year`. In reality, a mapping of date to all three of these arguments would be the best solution. We would therefore like to include *one* $\rightarrow$ *many, many* $\rightarrow$ *one* and *many* $\rightarrow$ *many* mappings in the algorithm.

Another way in which the algorithm could be improved is to make the scoring system more sophisticated. Currently, there is a single score assigned for mapping one node to another where one node is an abstraction of the other node. For example, the relationships between `tiger` and `feline` and `tiger` and `animal` are both abstraction relations, as `tiger` is a sub-class of both `feline` and `animal`, and they would therefore score the same. However, there is clearly a closer degree of kinship in the first relation than in the second, and a scoring system that could reflect this would provide a more accurate notion of similarity. In addition, allowing user-set weightings to affect the scoring would provide a much more accurate estimation of whether a service would perform a job satisfactorily. For example, if the constraint to be satisfied is:
`reading(RepID,Node,Date,Level),`
the services
`reading(RepID,Node,Date)` and
`reading(RepID,Node,Level)`
would both receive the same (low) score because they both omit an argument. However, perhaps the querier is very concerned to receive a value for `Level` but is not very bothered to receive a value for `date` (maybe the call is done in real time and the querier assumes that the date on which the reading is returned is the date on which it is made, so that this value becomes obsolete). In this case, we would like to allow the user to give a high weight to the `Level` argument and a low weight to the `Date` argument, meaning that the first mismatched service would score very low, whereas the second mismatched service would have quite a high score, reflecting that the fact that is could, despite mismatches, satisfy the querier.

These are the challenges we have currently identified with the SPSM algorithm; perhaps more will become apparent as the evaluation continues.

For the OpenKnowledge system as a whole, the largest challenge is to provide a complete and thorough evaluation done on a large scale. This is difficult due to the bootstrapping problem: proper evaluation depends on large numbers of services acting as OK peers in a natural and organic way – i.e., not set up by us solely for the purpose of evaluation. However, we cannot expect large numbers of services to become OK peers before we provide a full evaluation of the system. This problem is currently being made much more tractable as there are already many users of the OK system, and we intend to perform evaluation on their experience. Details of these early adopters can be found on the project webpage.

RUNNING THE OK SYSTEM

The full OpenKnowledge system, complete with full instructions and demonstrations, is available to download free from the project webpage[5]. Details of the emergency response testbed and simulator can also be found, together with complete documentation for the project. Once the full evaluation is completed, the results will be posted here as well as in the relevant publications.

CONCLUSIONS

The key contributions of this case are two-fold:

i) the introduction of the SPSM algorithm, which is broadly applicable and can be used in any situation where semantic tree-matching is necessary, making it applicable for service integration in most circumstances but also for many other forms of matching such as database integration;
ii) the introduction of the OpenKnoweldge system, which itself provides a major contribution to the problem of service integration by providing a complete framework in which this integration can occur, and also provides a demonstration of SPSM in action.

We have described a scenario in which both the full OpenKnowledge system and the SPSM algorithm have been demonstrated in action and evaluated. We briefly described an evaluation of the SPSM algorithm; further evaluation is currently taking place.

We believe that our approach offers a solution to a problematic and important issue: that of automatically integrating services in the many situations where hard-coding service calls is impractical or impossible. Whilst any solution to this problem that does not depend on a shared ontology must be an imperfect solution, it is nevertheless a solution that can be used in real-world, large-scale situations where the use of fully shared semantics is impossible.

Acknowledgements. We appreciate support from the OpenKnowledge European STREP
(FP6-027253).

References

van der Aalst, W.M.P., and ter Hofstede, A.H.M. (2005) YAWL: Yet Another Workflow Language. *Information Systems* 30(4) pages 245-275.
Aggarwal, R., Verma, K., Miller, J. A., and Milnor, W. (2004) Constraint driven web service composition in METEOR-S. In *Proceedings of IEEE SCC*.

[5] www.openk.org

Bergamaschi, S., Castano, S., and Vincini, M. (1999) Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1).

Besana, P., and Robertson, D. (2007) How service choreography statistics reduce the ontology mapping problem. In *Proceedings of ISWC*.

Ehrig, M., Staab, S., and Sure, Y. (2005) Bootstrapping ontology alignment methods with APFEL. In *Proceedings of ISWC.*

Euzenat, J., and Valtchev, P. (2004) Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*.

Euzenat, J., and Shvaiko, P. (2007) *Ontology matching*. Springer.

Fellbaum, C. (1998) *WordNet: an electronic lexical database*. MIT Press.

Giunchiglia, F., and Walsh T. (1989) Abstract theorem proving. In *Proceedings of "11th international joint conference on artificial intelligence (IJCAI'89)"*, pages 1372-1377, August 1989.

Giunchiglia, F., and Walsh, T. (1992) A theory of abstraction. *Artificial Intelligence*, 57(2-3).

Giunchiglia, F., Yatskevich, M., and Shvaiko, P. (2007) Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, IX.

Giunchiglia, F., McNeill, F., Yatskevich, M., Pane, J., Besana, P., Shvaiko, P. (2008a) Approximate Structure-Preserving Semantic Matching. In *Proceedings of "ODBASE 2008"*, Monterrey, Mexico, Nov 2008.

Giunchiglia, F., Sierra, C., McNeill, F., Osman, N., Siebes, R. (2008b) Deliverable 4.5: Good Enough Answers Algorithm. Techincal Report, OpenKnowledge. Retrieved November 2008 from www.openk.org.

Gligorov, R., Aleksovski, Z., ten Kate, W., and van Harmelen, F. (2005) Accurate and efficient html differencing. In *Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP),* pages 163–172. IEEE Press.

Gligorov, R., Aleksovski, Z., ten Kate, W., and van Harmelen, F. (2007) Using google distance to weight approximate ontology matches. In *Proceedings of WWW*.

Gooneratne, N., and Tari, Z. (2008) Matching independent global constraints for composite web services. In *In Proceedings of WWW,* pages 765–774.

Hu, W., and Qu, Y. (2006) Block matching for ontologies. In *Proceedings of ISWC*.

Kalfoglou, Y., and Schorlemmer, M. (2003) IF-Map: an ontology mapping method based on information flow theory. *Journal on Data Semantics,* I.

Klusch, M., Fries, B., and Sycara, K. (2006) Automated semantic web service discovery with OWLS- MX. In *Proceedings of AAMAS*.

Kotoulas, S., and Siebes, R. (2007) Deliverable 2.2: Adaptive routing in structured peer-to-peer overlays. Technical report, OpenKnowledge. Retrieved November 2008 from www.openk.org.

Li, L., and Horrocks, I. (2003) A software framework for matchmaking based on semantic web technology. In *Proceedings of WWW*.

Noy, N., and Musen, M. (2003) The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6).

Oundhakar, S., Verma, K., Sivashanugam, K., Sheth, A., and Miller, J. (2005) Discovery of web services in a multi-ontology and federated registry environment. *Journal of Web Services Research,* 2(3).

Robertson, D. (2004) A lightweight coordination calculus for agent systems. In

Declarative Agent Languages and Technologies, pages 183–197.
Shasha, D., and Zhang, K. (1997) Approximate tree pattern matching. In *In Pattern Matching Algorithms*, pages 341–371. Oxford University Press.
Shvaiko, P., and Euzenat, J. (2005) A survey of schema-based matching approaches. *Journal on Data Semantics*, IV.
Straccia, U., and Troncy, R. (2005) oMAP: Combining classifiers for aligning automatically OWL ontologies. In *Proceedings of WISE*.
Tai, K.-C. (1979) The tree-to-tree correction problem. *Journal of the ACM*, 26(3).
Tang, J., Li, J., Liang, B., Huang, X., Li, Y., and Wang, K. (2006) Using Bayesian decision for ontology mapping. *Journal of Web Semantics*, 4(1).