UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.disi.unitn.it

INTEGRATION OF SERVICES THROUGH STRUCTURED SEMANTIC MATCHING

Paolo Besana, Fiona McNeill, Fausto Giunchiglia, Lorenzino Vaccari, Gaia Trecarichi and Juan Pane

August 2008

Technical Report # DISI-08-038

Also: submitted to COOPIS 2008

Integration of Services through Structured Semantic Matching

Paolo Besana[1], Fiona McNeill[1], Fausto Giunchiglia[2], Lorenzino Vaccari[2], Gaia Trecharichi[2], Juan Pane[2]

[1] University of Edinburgh, Scotland,
p.besana@sms.ed.ac.uk|f.j.mcneill@ed.ac.uk
[2] University of Trento, Povo, Trento, Italy,
{fausto|vaccari|gtrecari|pane}@dit.unitn.it

Abstract The automated communication of services is crucial to the success of systems such as the Semantic Web. If global standards (the use of which is problematic) are not strictly adhered to, this requires services to be able to interpret both the vocabulary of calls made to them and the structure of these calls. In this paper, we describe the lifecycle of interaction within the OpenKnowledge system, which allows services to be found, contacted and interacted with during run-time without any prior agreement on semantics. Instrumental to this work is our *structure-preserving semantic matching* technique, which automatically matches inputs and outputs of services with calls representing service requirements, even if the vocabulary and structure of those calls are different to those expected by the service and unknown prior to run-time. We describe in detail a scenario showing the complexity of interaction allowed by our approach, and discuss the evaluation we have done on our techniques and the encouraging results this has produced.

1 Introduction

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 catagorisation of those terms is very context dependent. An ontology that included all terms that could be relevant to any situation would be infeasibly unwieldy and allow no flexibility for different interpretations of situations. However, this is not the only problem for communication of services. Calls to services are structured. For example, WSDL services require input and output messages, which are expected in a particular order and are defined within the WSDL file in a particular way. So for services to interact, there not only needs to be a process for interpretation of the different vocabularies but also a way to deal with the different structure of the calls.

It is perfectly possible to solve this problem by manually matching the expected inputs and outputs of two services prior to interaction. For example,

Altova [3] 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 services one will be required to interact with during run-time. 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 which it is possible to interact with.

In this paper, we propose a solution to this problem. Through focussing on shared workflows rather than shared semantic standards, we provide a way to allow the interaction of semantically heterogeneous services to successfully interact with one another. By not requiring shared semantics, we allow service designers to use language and structure that is appropriate to their world-view and the given context. Removing the need for shared semantics naturally introduces the need for matching between two different descriptions; however, through the use of these shared workflows, we reduce the vastness of the matching problem by providing a context for the interaction to take place in: only terms relevant to that context need be interpreted, and the context itself provides valuable information for the matching process. 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. Moreover, this matching not only detects perfect matches but can also identify *good enough* matches, where *good enough* is determined through changeable parameters [10]. This vastly increases the range of services it is possible to interact with.

These interactions take place within a system, the *OpenKnowledge*[4] framework [29], which provides the infrastructure necessary for the full life-cycle of this interaction process. That is, a discovery service is provided to find appropriate workflows for a given situation and, thereby, to find potentially suitable services with which to interact; a matching service (the focus of this paper) determines the similarity between requirement and ability; a trust component to allow users to assess with which services they wish to interact; and the infrastructure to facilitate these interactions is provided.

Section 2 introduces the language used to define interactions. Section 5 defines the steps the peers follows in order to select and execute a distributed interaction. Section 3 describes the matching process, and Section 4 shows one of the case studies used to evaluate the framework. Evaluation of our approach is given in Section 6, Section 7 presents related works and Section 8 concludes the paper.

2 Describing interactions

The core concept in OpenKnowledge are the interactions between participants, defined by *Interaction Models* (IMs) written in *Lightweight Coordination Calculus*: LCC [27,28] is a choreography language based on π-calculus and can be

[3] http://www.altova.com/

[4] `http://www.openk.org`

$$
\begin{aligned}
Model &:= \{Clause, \ldots\} \\
Clause &:= Role :: Def \\
Role &:= a\,(Type, Id) \\
Def &:= Role \mid Message \mid Def\ then\ Def \mid Def\ or\ Def \\
Message &:= M \Rightarrow Role \mid M \Rightarrow Role \leftarrow C \mid M \Leftarrow Role \mid C \leftarrow M \Leftarrow Role \\
C &:= Constant \mid P(Term, \ldots) \mid \neg C \mid C \wedge C \mid C \vee C \\
Type &:= Term \\
Id &:= Constant \mid Variable \\
M &:= Term \\
Term &:= Constant \mid Variable \mid P(Term, \ldots) \\
Constant &:= \text{lower case character sequence or number} \\
Variable &:= \text{upper case character sequence or number}
\end{aligned}
$$

Figure 1. LCC syntax

$$
\begin{aligned}
annotation &:= @annotation(about,\ innerAnnotation) \\
about &:= @role(Role) | @message(M) | @constraint(Term) | @variable(Variable) \\
innerAnnotation &:= annotation | tree \\
tree &:= Constant | tree | Constant, tree
\end{aligned}
$$

Figure 2. Annotations syntax

used as a compact way of representing distributed workflows. An IM in OpenKnowledge expresses both the *control-flow* and the *data-flow* perspectives of a workflow: it defines the activities that need to be performed and their order and specifies the flow of data between the components and the semantic structure of the data. Most orchestration-oriented workflow languages specify the behaviour of a single participant: for the other participants, only their invocation and their replies are defined. Nothing is said on their behaviour and on the interplay with the other actors: how they act and react to the unfolding messages they receive. The behaviour of the other participants is defined in separate, possibly unknown, workflows or it is embedded in their code. IMs in OpenKnowledge specify the behaviour of all the participants, focussing in particular on their interplay, expressed through the exchanged messages.

An IM in LCC is a declarative script, that is also directly executable using rewrite rules to expand the state and find the next move for each participant. An IM is a set of clauses, each of which defines how a role in the interaction must be performed. Roles are described by the type of role and an identifier for the individual participant undertaking that role. Depending on the IM, it may be possible for several participants to play the same role: for example, in the IM in Figure 7, we would expect severel participants to be playing the role of *fire_fighter* simulataneously (the number of possible participants for a single role is specified in the protocol). A single participant may also play several roles at once.

Participants in an interaction choose their *entry-role*. This is the role that they will initially play, which may entail playing subsequent, non-entry roles. For example, fulfilling the entry-role *seller* may entail taking on the role *deliverer*, or, in the IM in Figure 7, the entry-role *fire_fighter_ coordinator* leads to the role *fire_fighter_ coordinator(List)*. Participants follow the unfolding of the clause specified using a combinations of the sequence operator ('*then*') or choice operator ('*or*') to connect messages and changes of role. Messages are either outgoing to another participant in a given role ('$\Rightarrow$') or incoming from another participant in a given role ('$\Leftarrow$'). Message input/output or change of role is controlled by a constraint defined using the normal logical operators for conjunction, disjunction and negation. Constraints are expressed as first-order terms, and there is no explicit differentiation between inputs and outputs. For example, the constraint $get\_route(From, To, Path)$ shown in the interaction model in Figure 7 finds a path between two points: the input parameters are *From* and *To*, while the output parameter is *Path*. The input parameters must be already instantiated when the constraint is called, while the output parameters are instantiated through the solving of the constraint. There is no commitment to the method used to solve constraints - so different participants might operate different constraint solvers (including human intervention). Figure 1 defines the syntax of LCC.

An IM specifies only the abstract exchange of messages between participants, and the fact that some condition must hold before or after these messages: it cannot specify the type of the parameters in messages or constraints, or timeouts. This means that the same interaction model can be used in different contexts,

where the conceptual workflow remains the same, but only the content of messages changes.

However, the framework relies on matching IMs with participants' components, which requires semantic information about both, and real world applications may need to express timeouts or additional informations about constraints or messages. To enable this, we added a layer of annotations, whose syntax is shown in Figure 2. Any element in the IM can be annotated: variables, messages, roles, constraints. The only annotations whose meaning is specified and used inside the framework are those about variables, that define their semantic structures. The other annotations are open to future applications: different community will use them differently. The framework provides access to them, but its behaviour can be extended.

The semantic annotation of variables currently uses keywords and trees of keywords to represent structured parameters. The scope of a variable is a role clause, so variable annotations are inside role annotations. Figure 3 shows an example of annotation: the variable `Event`, within the role `alarmClock`, is a tree structure whose nodes are the name, the description and the date of the event. The date is a tree structure composed by year, month and day. Using tags simplifies the work of the developers, and makes the code more readable but is less stringent: it is possible to replace tags with URI from formal ontologies.

In the OpenKnowledge system, we expect that only a minority of users would be interested in writing their own IMs. Generally, users will search for and resuse those IMs that have been written by this core group. For those that wish to write their own IMs or to alter existing IMs to better suit their purposes, we provide tools to facilitate this. However, IMs are intented to be general and reusable and the majority of users are free to ignore the technical detail discussed here.

```
a(alarmClock,A)::
alert(Event)=>a(recipient,R) <- importantAlert(Event)
...
@annotation(@role(alarmClock),
    @annotation(@variable(Event),event(name,description,date(year,month,day))))
```

Figure 3. Example of annotated variable

The IMs are published by the authors on the *distributed discovery service* [22] with a keyword-based description. The roles in the IMs are played by *peers*: a peer is a node in the network that is able to perform the basic activities of subscribing to a role in an IM and satisfying the constraints in that role. 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. A plug-in component is `jar` file that provide an interface to a set of annotated methods [3]. The methods can be simple wrappers for web services. We have developed a tool that generates a wrapper component from a WSDL file: each operation in it becomes a method in the component. The peer can be a

GUI-based application whose components interact directly with a user, such as Skype, or a server application that solves the constraints automatically, possibly calling the web services wrapped by the components or accessing a database.

Interactions are run in order to perform some task that requires the coordination of various participants. Assuming that an interaction model for the task has already been published, when the need for such a task arises, the lifecycle of an interaction is:

***Interaction selection*:** a peer that wants to perform some task searches on the discovery service for the published IMs for the task by sending a keyword query. The discovery service 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: Section 3 describes the process in detail. If it finds a suitable IM, it subscribes on the discovery service to perform the appropriate role in it. For example, in the scenario described in Section 4, firefighters subscribe to play the role *fire_fighter* and various route finders (which may belong to different organisations or be independent) subscribe to the role *route_ service*. When the fire coordinator needs to tell the firefighter to go somewhere, it look up this IM, tries to match its components with the constraints and then subscribes to it. In another scenario, such as a buyer/seller one, various vendors may be subscribed to different purchase IMs: when a customer needs to buy something, it look up for the purchase IMs and then subscribes to the one the best fits its component.

***bootstrap*:** when all the roles in an IM are subscribed, the discovery service randomly selects a peer in the network, asking it to play the coordinator of the interaction. This peer may or may not be subscribed to play a role in the IM - though if the peer network is large, this would be very unlikely. If it accepts, it becomes the IM coordinator (not to be confused with any coordination role within the IM, such as the fire coordinator) and asks all the subscribed peers to select who they want to interact with. The peers may use different mechanisms to select the peers: they can use their own past experience, or access a shared repository of experiences, or ask other peers. The OpenKnowledge system provides a trust component to assist peer in making this assessment [10]. After receiving the peers' preferences, the IM coordinator creates a group of peers who are all happy to interact with one another in their proposed roles. If the group covers all the roles, it starts the interaction. In the emergency response scenario, when the fire coordinator subscribes to the IM in Figure 7, the discovery services checks whether all the roles are subscribed. If so, the discovery service can delegate a random peer to be the IM coordinator and bootstrap the interaction. All peers subscribed receive the list of the subscriptions. The emergency coordinator may have an internal list of firefighter to contact, and choose to interact only with them. In the purchase scenario, the customer may want to interact only with a specific vendor as he trust it. The vendor, on the other hand, may reject the customer, who may be on his own black list.

***run of the interaction*:** the IM coordinator creates a proxy for every peer in the IM, and runs it locally. The messages are exchanged between the proxies, while the peers are contacted in order to solve the constraints. In the example in Figure 9, the *firefighter1* is called to solve the constraint $getPos(Pos)$. If the peer application runs on the firefighter's palm device, the method solving the constraint may access the intenal GPS, or beep and ask the person to introduce the information through a GUI.

***follow-up*:** after the run of the interaction, the IM coordinator sends the log of the interaction to all the involved peer so that they can analyse it. The analysis can be aimed at computing a trust value for the other peers [10] 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 [4]. For example, after the interaction in Figure 7 has run, a firefighter may find that the path provided by the route-finder is blocked and report somewhere (for example on a reputation server) that the route service has been unreliable. If, interaction after interaction, the route service is consistently unreliable, it will be selected less and less 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.

3 Matching service descriptions

One of the key feature of the framework is its inherent capability of handling heterogeneity dynamically. A peer can download a plug-in component written for a slightly different IM than the one in which the peer wants to participate: the framework tries to match the methods in the components with the constraints, creating an adaptor for transparently accessing the parameters from within the method. Even in situations where the interactions and the components where agreed in advance, they may drift over time.

The methods, as briefly stated above, are annotated in a similar way to the constraints (using Java 5 annotation mechanism): the arguments can be structures, and the elements in the structures are accessed using their path. Figure 4 shows an example of a method that would be matched by the framework to the constraint "`importantAlert(Event)`" in Figure 3, obtaining the correspondences in Figure 5.

Internally, the method accesses the elements of its arguments using the locally defined structure: so to access the day of the date of the event, it will use parameter D, asking for the value of "date/day", independently of how it was

```
class Component extends OKCFacade{
 ...
 @MethodSemantic(language="tag",
 args={"event(name_of_event,comment)", "date(day,month,year))"}
 boolean relevantAlert(Argument E, Argument D){...}
 ...
}
```

Figure 4. Example of annotated method

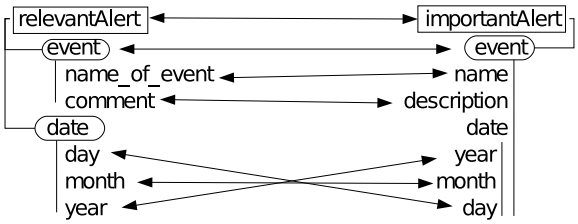

Figure 5. Matching between the constraint in Figure 3 and the method in Figure 4.

defined in the IM and how it was used by the other peers: the access and the storage of the arguments are decoupled by an adaptor.

If a component wraps a WSDL file, each method corresponds to an operation in WSDL, and the method annotations reflect the structures defined in it: thanks to the adaptor, accessing the web service is decoupled from the representation of the constraint in the IM.

The matching process is organized in two steps: (i) node matching and (ii) tree matching. Node matching solves the semantic heterogeneity problem by considering only labels at nodes and contextual information of the trees. We use here the S-Match system [17]. Technically, two nodes $n_1 \in T1$ and $n_2 \in T2$ match iff: $c@n_1\ R\ c@n_2$ holds, where $c@n_1$ and $c@n_2$ are the concepts at nodes n_1 and n_2, and $R \in \{=, \sqsubseteq, \sqsupseteq\}$. In semantic matching [13,14,9] as implemented in the S-Match system [17] the key idea is that the relations, e.g., equivalence and subsumption, between nodes are determined by (i) expressing the entities of the ontologies as logical formulas and by (ii) reducing the matching problem to a logical validity problem. Specifically, the entities are translated into logical formulas which explicitly express the concept descriptions as encoded in the ontology structure and in external resources, such as WordNet [8]. This allows for a translation of the matching problem into a logical validity problem, which can then be efficiently resolved using sound and complete state-of-the-art satisfiability solvers [15]. Note that the result of this stage is the set of one-to-many correspondences holding between the nodes of the trees.

Tree matching, in turn, exploits the results of the node matching and the structure of the trees to find if these globally match each other. Specifically, given the correspondences produced by the node matching, abstraction operations are used [12,11] in order to select only those correspondences that pre-

serve the desired properties, namely that functions are matched to functions and variables to variables. Then, the preserved correspondences are used as allowed operations of a tree edit distance in order to determine global similarity between trees under consideration. If this global similarity measure is higher than an empirically established threshold, the trees are considered to be similar enough, and are considered to be not similar otherwise. Technically, two trees $T1$ and $T2$ approximately match iff there is at least one node n_{1i} in $T1$ and a node n_{2j} in $T2$ such that: (i) n_{1i} approximately matches n_{2j}, and (ii) all ancestors of n_{1i} are approximately matched to the ancestors of n_{2j}, where i=1,...,N1; j=1,...,N2; N1 and N2 are the number of nodes in $T1$ and $T2$, respectively.

Semantic heterogeneity is therefore reduced in two steps: (i) matching the web services, thereby obtaininging an alignment, and (ii) using this alignment for the actual web service integration. This process is discussed in detail in [16].

4 Case study

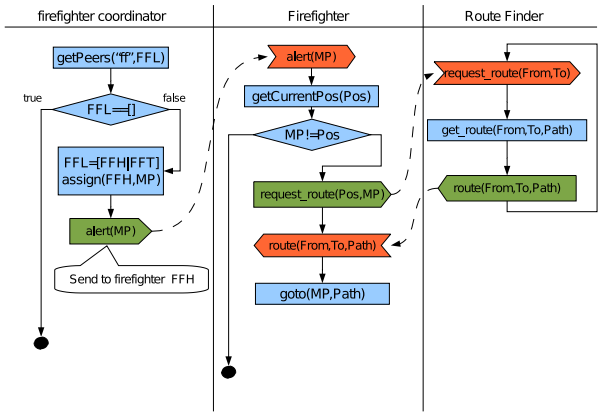

Figure 6. Activity diagram for the interaction

5 Life cycle

The Open Knowledge system is fully general and can be applied in any domain in which systems (or peers, services, processes, etc) are interacting. However, it has been specifically evaluated in two testbeds, one of which is emergency response. This was chosen as being a particularly knowledge-intensive and dynamic application domain, with many players and a high potential for unexpected developments.

$$
\begin{array}{l}
a(fire\_fighter\_coordinator, FFC) :: \\
\quad null \leftarrow getPeers(\text{“}fire\_fighter\text{”}, FFL) \\
\quad then\ a(fire\_fighter\_coordinator(FFL), FFC)
\end{array}
$$

$$
\begin{array}{l}
a(fire\_fighter\_coordinator(FFL), FFC) :: \\
\quad null \leftarrow FFL = [] \\
\quad or \left(\begin{array}{l} alert(MP) \Rightarrow a(fire\_fighter, FFH) \leftarrow FFL = [FFH|FFT]\ and\ assign\_mp(FFH, MP) \\ then \\ a(fire\_fighter\_coordinator(FFT), FFC) \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
a(fire\_fighter, FF) :: \\
\quad alert(MP) \Leftarrow (fire\_fighter\_coordinator, FFC) \\
\quad then\ \ null \leftarrow getPos(Pos) \\
\quad then\ \ \ (null \leftarrow equal(MP, Pos)) \\
\quad or \left(\begin{array}{l} request\_route(Pos, MP) \Rightarrow a(route\_service, RS) \\ then\ \ route(From, To, Path) \Leftarrow a(route\_service, RS) \\ then\ \ null \leftarrow goto(MP, Path) \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
a(route\_service, RS) :: \\
\quad request\_route(From, To) \Leftarrow a(fire\_fighter, FF) \\
\quad then\ \ route(From, To, Path) \Rightarrow a(fire\_fighter, FF) \leftarrow get\_route(From, To, Path) \\
\quad then\ a(route\_service, RS)
\end{array}
$$

Figure 7. IM for the e-rescue interaction

$$
\begin{array}{l}
@annotation\,(@role\,(route\_finder)\,, \\
\quad @annotation\,(@variable(From),\ from(location(name, lat, long)))) \\
@annotation\,(@role\,(route\_finder)\,, \\
\quad @annotation\,(@variable(To),\ to\,(location\,(name, lat, long)))) \\
@annotation\,(@role\,(route\_finder)\,, \\
\quad @annotation\,(@variable(Path),\ path\,(list\,(location\,(name, lat, long)))))
\end{array}
$$

Figure 8. Semantic annotations relative to the interaction

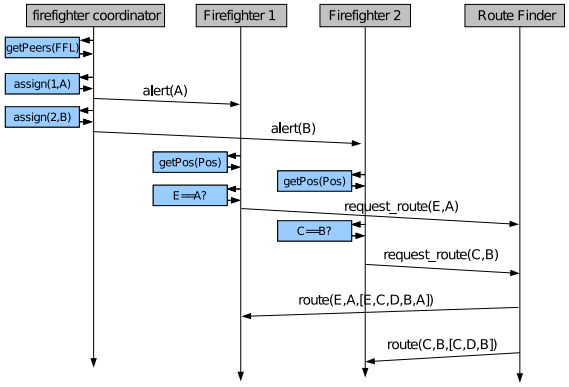

Figure 9. A possible run of the IM in Figure

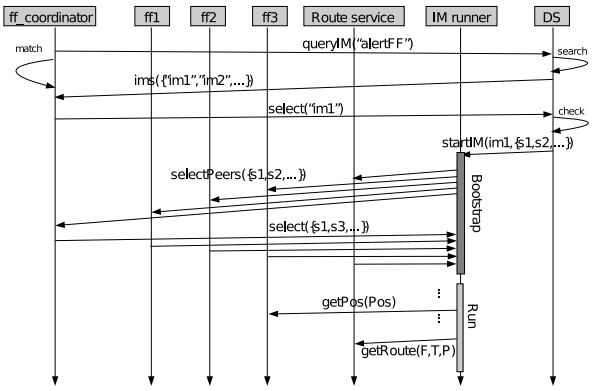

Figure 10. Lifecycle for the IM used in the scenario

In this section, 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 flooded 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 who 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, service may develop and change and it is unrealistic to expect these changes would always be known and accounted for in advance.

The particular interaction we focus on here is the fire control centre sending its fire teams to destinations where they are needed. The IM for this interaction is shown in Figure 7 and Figure 8 shows the annotations for this IM. Figure 6 illustrates this interaction as an activity diagram and Figure 9 represents a possible run of this interaction with two fire teams. The lifecycle of the interaction interaction process is illustrated in Figure 10

The interaction is as follows:

- The fire coordination centre receives an emergency warning from the flood monitoring centre and details of the current state of the flood, together with information about vulnerable people and goods. The fire coordination centre then works out the places where the fire teams should be sent to (this part of the process is not described in the diagrams or the IM).
- The fire coordination centre uses the routing service to locate the teams that are subscribed to playing the role of fire team. Of course, we may assume that the fire coordination centre knows who the fire teams are, so this routing process may not be necessary, but the advantage is that it provides information about who is available to play their role at any given time.
- The fire coordination centre circulates one of the chosen destinations to each of the fire teams. In this particular scenario, this is done arbitrarily with no reference to where the fire teams currently are.
- Once a fire team receives its destination, it will first check whether it is already at that destination and, if not, it will make plans to move there. This involves moving on to a new part of the interaction, this time involving a routing service, to find out how it should get to the chosen destination.
- The routing service is able to provide a route between two points that takes into account roads that might be flooded, blocked or otherwise inaccesible. It does this by having a map of the area and through running separate interactions with sensor nodes and with other peers to keep track of the current state of the roads.
- Once the fire teams have a route, they move to their destination and the interaction terminates.

We would expect, since these interactions represent the expected protocol of a flooding response, that the fire teams would already know a routing service and would have practiced these interactions. However, there are nevertheless reasons why call to such a routing service may be inappropriate. For example, the routing service may have altered over time - the route-service peer may be involved frequently in many other interactions and its components may be changed over time to adapt better to these interactions - and have failed to keep the fire service up to date with these changes (a service would not normally expect to keep all service users up to date), meaning that the call the fire service makes would be incorrect and matching would be required for the interaction to succeed. Other situations could be that the routing service lost connection or was swamped with requests, and the fire team would then have to use the discovery service to locate a new routing service and reuse the old routing service call to call this new routing service.

6 Evaluation

We have seen that drifting can cause heterogeneity between components and IMs: components that were designed for a particular interaction are used in others and change over time to adapt to these: when they are the reused in the

1. *replace a label with an unrelated label*: a node label can be replaced with an unrelated label. The unrelated label is randomly selected from a dictionary.
   Example:
   - Original tree: `find_Address_By_Point(point, address_Finder_Options, part)`
   - Altered tree: `find_Address_By_Point( atom_firmer, discussion, part)`
2. *add or remove a term in a node label.*
   Example:
   - Original tree: `find_Address_By_Point(point,address_Finder_Options, part)`
   - Modified tree: `find_By_Point(toast_point, address_Options, surface_part)`
3. *replace a term in a node label with a related one*: e.g., by using synonyms, hyponyms, hypernyms from WordNet 3.0.
   Example:
   - Original tree: `find_Address_By_Point(point,address_Finder_Options, part)`
   - Modified tree: `find_Address_By_Point(aim,address_Finder_Options, section)`
4. *alter syntactically a label*: characters in the label are dropped, added, or changed.
   Example:
   - Original tree: `find_Address_By_Point(point,address_Finder_Options, part)`
   - Modified tree: `finm_Address_By_Poioat(einqt, ddress_Finder_Optxions, vparc)`

Table 1. Tree alterations applied to services's signatures

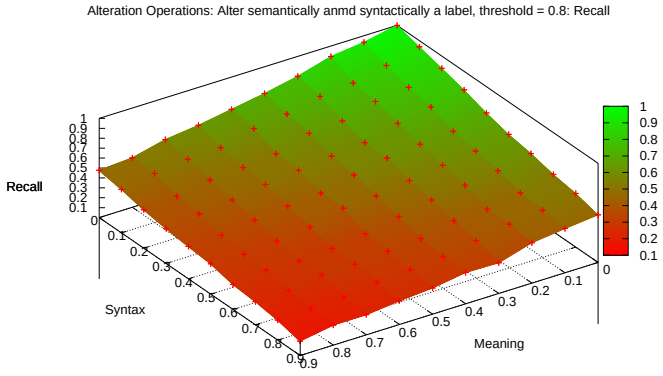

Figure 11. Recall for increasing different functions

original interaction, matching is required. Similarly, interactions designed for a specific context may be used for different aims and change to adapt. Moreover, new interactions or components can be developed by copying others.

Starting from these assumption, we tried to evaluate how the matching mechanism, described in Section 3, can cope with these sort of heterogeneity.

Thus, the evaluation dataset was composed of trees that are alterations of several original trees. Initially, 80 trees were built out of the ESRI ArcWeb Services (SOAP methods[5]). Examples include: `find_Address_By_Point(point, address_Finder_Options,part)`, `get_Distance(location1, location2, num_Points, return_Geometry, token, units)` and `convert_Map_Coords_To_Pixel_Coords(map_Area, map_Size, map_Coords, token)`. Then, for each of the 80 original trees, 20 altered ones were automatically generated. Pairs were composed of the original tree and one varied tree, thereby resulting in 1600 matching tasks, which were then matched using our structure-preserving semantic matching techniques. The tree alteration procedure has been inspired by the work in Euzenat [6] on systematic benchmarks. The alteration operations, semantic or syntactic, are applied to nodes and are shown in Table 1.

Since the tree alterations made are known, these provide the ground truth, and hence the reference results are available for free by construction: 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 probabilitistically 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.

Figure 11 shows how recall behaves when the probabilities of syntactic and semantic alterations are increased. Recall decreases slowly: only when both se-

[5] http://www.arcwebservices.com/v2006/help/index.htm

mantic 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.

7 Related work

We believe that this approach to structured matching is unique and therefore it is difficult to perform any comparitive analysis. In order to demonstrate that we make use of powerful ontology matching tools for the standard ontology matching step of the process, we can compare S-Match against other ontology matching tools. However, the full structure preserving matching addresses a previously unsolved problem. In this section, we discuss other methods that address similar problems.

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 a 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 [21,23,26] for example. Probably the most similar to ours is the approach taken in METEOR-S [1] and in [25], 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 context provided by the first order term.

Many diverse solutions to the ontology matching problem have been proposed so far. See [30] for a comprehensive survey and [7,24,5,18,2,20,31] 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) [32]. The approach taken in [19] 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 first order terms.

8 Conclusion

Successful service integration is fundamental to the realisation of the semantic web. The common approach is to write workflows that integrate service calls, creating a new, more sophisticated service. The approach we have presented in this paper, based on the OpenKnowledge project, relies on shared distributed workflows, called Interaction Models. We have described the lifecycle of an interaction within the OpenKnowledge system, which allows peers - which may

be services or processes of any kind - to interact with one another. This utilises our structure-preserving semantic matching technique that allows us to map the calls to services from these workflows with the inputs and outputs expected by the service, even when these are unknown prior to run-time. In this way, we facilitate the automated interaction of services without any agreed semantics or any expectations prior to run-time of what service we will interact with. We have illustrated this process through a case study based on emergency response and have presented promising results indicating the efficacy of this approach.

References

1. R. Aggarwal, K. Verma, J. A. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *Proceedings of IEEE SCC*, 2004.
2. S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1), 1999.
3. P. Besana, D. Dupplaw, and A. De Pinnick. Openknowledge deliverable 1.3: Plug-in components. Technical report, 2007.
4. P. Besana and D. Robertson. How service choreography statistics reduce the ontology mapping problem. In *ISWC2007*, 2007.
5. M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with APFEL. In *Proceedings of ISWC*, 2005.
6. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, 2007.
7. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*, 2004.
8. C. Fellbaum. *WordNet: an electronic lexical database*. MIT Press, 1998.
9. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. In Meersman R. and Tari Z., editors, *On the move to meaningful internet systems 2005: coopIS, DOA, and ODBASE: OTM Confederated International Conferences*, volume 3760/2005 of *LNCS*, Agia Napa, Cyprus, 2005.
10. F. Giunchiglia, C. Sierra, F. McNeill, N. Osman, and R. Siebes. Deliverable 4.5: Good enough answer algorithms. Technical report, University of Edinburgh, 2008.
11. F. Giunchiglia and T. Walsh. Abstract theorem proving. In Bassi S., Sridharan N., Bertacco S., and Bonicelli R., editors, *11th international joint conference on artificial intelligence (IJCAI'89)*, volume 1, Detroit, Mich., 20-25 August 1989 1989.
12. F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3), 1992.
13. F. Giunchiglia and M. Yatskevich. Element level semantic matching. In Mcilraith S. A., Plexousakis D., and Van Harmelen F., editors, *The Semantic Web: ISWC 2004: Third International Semantic Web Conference: Proceedings*, Hiroshima, Japan, 6 November 2004 2004. Berlin: Springer.
14. F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In Gomez perez A. and Euzenat J., editors, *Proceedings of the 2nd European semantic web conference (ESWC'05)*, volume 3532/2005 of *Lecture Notes in Computer Science*, Heraklion, Crete, Greece, 29 May - 1 June 2005 2005.
15. F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *Proceedings of ESWC*, 2005.

16. F. Giunchiglia, M. Yatskevich, and F. McNeill. Structure preserving semantic matching. In *Proceedings of the ISWC workshop on Ontology Matching*, Busan, Korea, November, 2007 2007.
17. F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, IX, 2007.
18. R. Gligorov, Z. Aleksovski, W. ten Kate, and F. van Harmelen. Using google distance to weight approximate ontology matches. In *Proceedings of WWW*, 2007.
19. W. Hu and Y. Qu. Block matching for ontologies. In *Proceedings of ISWC*, 2006.
20. Y. Kalfoglou and M. Schorlemmer. IF-Map: an ontology mapping method based on information flow theory. *Journal on Data Semantics*, I, 2003.
21. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *Proceedings of AAMAS*, 2006.
22. S. Kotoulas and R Siebes. Deliverable 2.2: Adaptive routing in structured peer-to-peer overlays. Technical report, OpenKnowledge.
23. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of WWW*, 2003.
24. N. Noy and M. Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6), 2003.
25. S. Oundhakar, K. Verma, K. Sivashanugam, A. Sheth, and J. Miller. Discovery of web services in a multi-ontology and federated registry environment. *Journal of Web Services Research*, 2(3), 2005.
26. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of ISWC*, 2002.
27. D Robertson. A lightweight coordination calculus for agent systems. In *Declarative Agent Languages and Technologies*, pages 183–197, 2004.
28. D Robertson. A lightweight method for coordination of agent oriented web services. In American Association for Artificial Intelligence, editor, *Technical Report SS-04-06*, 2004.
29. D. Robertson, F. Giunchiglia, F. van Harmelen, M. Marchese, M. Sabou, M. Schorlemmer, N. Shadbolt, R. Siebes, C. Sierra, C. Walton, S. Dasmahapatra, D. Dupplaw, P. Lewis, M. Yatskevich, S. Kotoulas, A. P. de Pinninck, and A. Loizou. Open knowledge semantic webs through peer-to-peer interaction. Technical Report DIT-06-034, University of Trento, 2006.
30. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics*, IV, 2005.
31. U. Straccia and R. Troncy. oMAP: Combining classifiers for aligning automatically OWL ontologies. In *Proceedings of WISE*, 2005.
32. J. Tang, J. Li, B. Liang, X. Huang, Y. Li, and K. Wang. Using Bayesian decision for ontology mapping. *Journal of Web Semantics*, 4(1), 2006.