Facilitating Interaction Between Virtual Agents
Through Negotiation over Ontology Representation

Fiona McNeill
Heriot-Watt University
f.mcneill@hw.ac.uk

BACKGROUND

It is universally acknowledged that the problem of integration of information across large communities is a difficult and pressing one, particularly when these communities are disparate, widespread and not under centralised control, such as in the Semantic Web (Berners-Lee et al, 2001). The simplest solution to this problem is the enforcement of a single ontology: a single view of the world. However, if the agents interacting are from different organisations or fields, attempting to use a single ontology is usually neither practical nor desirable. Users need to develop a representation that is best suited to their own problems and they need to maintain and update that representation locally. Even if all users do subscribe to a single ontology, integration problems still exist, as changes and updates are made and users tune their ontologies to fit their own needs.

The problem of ontology matching has been widely studied and powerful solutions are available (see (Shvaiko and Euzenat, 2013) and (Euzenat and Shvaiko, 2013) for a comprehensive survey). However, the ontologies considered are almost always taxonomies, and the problem of ontology matching is concerned with relating a single term in one ontology to one or more terms in another ontology: for example, a term car in one ontology may relate to a term automobile or a term carriage in another ontology. Much less considered is the problem of relating compound terms such as first-order terms or database entries with multiple fields: for example, a term car(make,model) in one ontology relating to a term automobile(model,year,brand) in another. In such situations we still have the problem of relating the single terms contained within these compound terms – e.g., this matching depends on knowing that car may be related to automobile and that make may be related to brand. But we must also consider the overall relation of the compound terms, which requires not only semantic but also structural matching.

Another drawback of traditional ontology matching in an online environment is that it tends to assume full knowledge of all relevant ontologies and is generally performed off-line, prior to interaction. These are the assumptions made by the main evaluation processes for Ontology Matching, such as the Ontology Alignment Evaluation Initiative (OAEI1). But in large, fast-moving agent communities, or situations where some information may be confidential, we cannot assume that we can have full knowledge of any agent or service we may interact with, nor is it possible to perform the matching off-line if we may not know prior to interaction which agents will need to interact.

In this paper, we introduce our theory of on-the-fly, structured matching and briefly describe the ORS system, which we have developed to implement this theory. Our central hypothesis is that representation – as well as vocabulary and beliefs – must be treated as a fluent and that automated, dynamic matching techniques that can map between

1 http://oaei.ontologymatching.org/
structured terms are necessary for full integration of disparate ontologies (Bundy et al., 2006).

FOCUS OF APPROACH

In a system such as the Semantic Web, where there is no centralised control, we cannot have a complete global overview of the agents and data in the system. Agents may join and leave the system freely and they will all have their own ontologies and data that may be large and complex and may be confidential. We cannot hope for a complete description of the relations between every agent in the system. Our approach is therefore not to consider how such a system can be controlled but how an individual agent can successfully make its way in such a system, interacting with the agents that it needs to interact with, even if these agents are not using the same ontological terms or representations, and even if it is not known in advance of the interaction which agents these will be.

Although many existing ontologies are simple taxonomies, and matching these ontologies is a crucial task, we believe that this kind of matching cannot be sufficient. Agents that are capable of interacting in complex and unpredictable environments need to be able to plan, and planning agents need far richer descriptions of the world: not only taxonomies of classes but also relations and functions between these classes, and planning rules describing how to influence the world. Uniformity of these relations, functions and rules can no more be assumed than uniformity of terms within taxonomies, and therefore matching between these structured objects is just as crucial as the more frequently addressed problem of matching between simple terms within a taxonomy.

Matching large ontologies at run-time, particular ones that contain structured terms, is generally not feasible, but we make this problem tractable by only fixing mismatches when this is demonstrated to be necessary. Since interactions may be frequent and fleeting, there may not be much value in matching the full ontologies, since the interaction that is desired may only require a very small part of the ontologies, and even if inconsistencies exist between the ontologies, these may not lead to communication breakdown between agents during a particular interaction. Our approach is therefore to diagnose mismatches and refine the ontologies accordingly only when these mismatches directly cause communication breakdown.

To this end, we developed ORS (the Ontology Repair System)

2. This is a tool that an individual agent (which we name PA – the planning agent) can make use of as an aid when communication breaks down. ORS tracks the course of the communication between PA and any agents it may be interacting with (we name these SA – service-providing agents). If communication proceeds successfully then ORS does not need to be utilised. However, if communication breaks down, ORS begins the diagnostic process, analysing the communication so far and prompting PA to ask further questions in order to pin down a specific mismatch between the ontologies of PA and SA, which is then corrected.

The benefit of ORS is therefore that it allows an agent to interact successfully with other agents, even when their ontologies are mismatched in important ways, and even when this mismatch is between complex, structured ontological objects as well as when this mismatch is between simple terms. It works on the fly and fully automatically even when

http://dream.inf.ed.ac.uk/projects/dor/
interactions are unpredictable and unforeseen.

**ONTOLOGICAL MISMATCH**

Planning agents require ontologies that contain three different kinds of objects, which entail three different kinds of mismatches:

1. purely semantic mismatch, where the mismatch is between words or phrases – for example, *car* is matched to *auto*: this is the problem that is covered by conventional ontology matching.

2. structural mismatch, where the mismatch is between structured terms (such as relations and functions) – for example, *car*(Make,Model) is matched to *car*(Make), or *car*(Make,Model,Year) is matched to *car*(Make,Model,Date).

3. Mismatches of planning rules, where one agent has a different idea of the conditions and effects of performing an action to another – for example, *BuyTicketAction: wants_to_travel(Me,Destination) → has_ticket(Me,Destination)* matched to *BuyTicketAction: wants_to_travel(Me,Destination) and has_money(Me) → has_ticket(Me,Destination)*.

Of these, the first point is only considered incidentally, due to the large body of work that already addresses this issue; our emphasis is strongly on the second and third points.

**Structured Matching**

The problem of structural mismatch within ontological mismatch is crucial to successful interaction of agents or services. Not only are the utterances of agents usually structured, but service invocations are also necessarily structured, and their automatic interaction requires structured matching of just the type our work addresses; therefore, semantic matching alone cannot be sufficient.

In our work, we consider quantifier-free, first-order terms; that is, predicates with some number of arguments ≥ 0. Most common service invocations, such as those expressed in BPEL, as well as most types of database entries, can be expressed in such a way. Our techniques are therefore very widely applicable.

The space of possible mismatches between one first-order term and another, more general, first-order term is described by the theory of abstraction (Giunchiglia and Walsh, 1992). They describe four kinds of mismatch:

- **Propositional abstraction:** A term is matched to one with fewer arguments – for example, *car*(Make,Model) maps to *car*(Make).

---

3 Note that the → in these rules implies the performance of an action: the predicates on the left-hand side must be true before the action is performed; the predicates on the right hand side are made true after the action is performed.

**Predicate abstraction:** A term is matched to one with a more general predicate – for example, `car(Make)` maps to `vehicle(Make)`.

**Domain abstraction:** A term is matched to one with a more general type of argument – for example, `car(Make,Second-hand-dealer)` maps to `car(Make,Dealer)`.

**Precondition abstraction:** A term is removed from a rule – for example, `has(money,Me) → owns(car(Make),Me)` maps to `has(money,Me) & has(id,Me) → owns(car(Make),Me)`.

By inverting these relationships, we obtain four dual refinement operators. These abstraction and refinement operations are sufficient for describing most ways in which quantifier-free, first-order terms can be related. Non-identical terms must be either synonymous, more general, less general or unrelated. The case where they are synonymous is dealt with by semantic matching. There are a limited number of attributes of first-order terms: they have predicate names, numbers of arguments, orders of arguments and types of arguments. Variations in these attributes gives the above classification of how first-order terms may be mismatched.

**WORKED EXAMPLE**

Consider the interaction between a planning agent (PA) which wishes to buy a ticket, and an agent acting as the front-end to a ticket-selling service (the service providing agent – SA). PA contacts SA with the following message:

PA: `buy(pa,london,edinburgh)`

indicating that PA wishes to buy a ticket between London and Edinburgh. It is likely that SA will have some conditions on selling tickets, for example that the buyer has money, and must verify that these conditions hold. SA will therefore respond with a question:

SA: `money(pa,Amount)`?

This indicates that SA wishes PA to find a suitable instantiation of the variable `Amount` such that the relation `money(pa,Amount)` becomes true. But imagine that PA has money represented as a predicate `money(Agent,Amount,Currency)`, and can instantiate this to `money(pa,100,dollars)`. Perhaps this agent is used to operating in many different currency zones, whereas SA only operates in one currency zone, so their designers chose different representations. This question from the SA is considered a *surprising question* because it does not directly match any of the preconditions in the PA's rule.

PA must therefore respond negatively to SA's enquiry:

PA: `no`
SA: `fail: buy(pa,london,edinburgh)`

However, PA is able to analyse the problem and alter its ontology appropriately. In this

---

5 Note that we use a lower case initial letter to indicate a constant and an upper case initial letter to indicate a variable. In theory, the name of the variable can be anything, but we use the convention of using the type name so that the type information is immediately apparent. Thus `money(pa,Amount)` indicates that the predicate `money` relates a specific agent named `pa` has an unknown value which must be of type `Amount`. 
instance, it is obvious that the problem is that $money/2$ is mismatched to $money/3$. PA can determine that the two arguments of SA's $money/2$ match the first two arguments of its $money/3$, and hence the third argument – Currency – is redundant. It therefore alters its definition of $money$ from $money(Agent,Amount,Currency)$ to $money(Agent,Amount)$, and alters all instantiations of this predicate accordingly. This means that it is removing information (that the currency is dollars) that may be important in subsequent interactions. This could be resolved by retaining a copy of the old $money/3$, whilst using the updated $money/2$ in the current interaction. Additionally, it is making the assumption that the $Amount$ referred to by SA is in dollars. If the assumption in SA's design that money is in dollars is implicit, it cannot be accessed automatically. Confirmation of this would therefore involve user intervention.

The interaction can then be resumed and this time PA will be able to respond appropriately.

(Note that several important steps, such as discussion of price, have been omitted from this simplified example).

THE ONTOLOGY REPAIR SYSTEM

In this section, we describe the role of ORS through giving a step-by-step description of how it is used by an agent (PA) within a multi-agent, semantic-web-like environment.

1. PA has a goal, or is provided by the user with a goal, and uses its ontology to form a plan to achieve this goal. The steps of this plan will normally involve interaction with other agents. For example, if PA has to organise a journey, it will need to interact with (among others) a ticket-selling agent. Each of PA's planning rules will describe the circumstances under which it believes these other agents (SAs) will perform their roles (e.g., a ticket-selling agent will sell a ticket if it is provided with an appropriate destination and sufficient money). If there are no mismatches, the PA's rule will be compatible with the SA's rule which the SA is actually using to perform the task.

2. In order to achieve the goal, PA will execute each step by communicating with the appropriate SA.

3. PA will request the service it requires from the agent, and then wait for a response. This could be an indication that the service has been performed, but it will usually be further questioning from the SA, which is trying to establish whether or not the preconditions are satisfied, to confirm that it can perform the service.

4. If this process results in the service being successfully performed, PA proceeds to the next step of the plan and the functionality of ORS is not required at this step. However, if failure occurs, PA invokes ORS to attempt to track down the cause of this failure.

5. ORS will analyse all the questioning that has occurred so far, giving particular attention to questions the SA asked that were not expected. Expected questions
would tie in with the preconditions of PA's rule for the relevant service; any questions deviating from these, either substantially or through a structural difference, would be a surprising question (SQ). An example from the previous section is the question `money(pa,Amount)` being asked when the question `money(pa,Amount,dollars)` was expected; alternatively, a completely unexpected question might be asked.

6. ORS's diagnostic algorithms (briefly described below) use this information, prompting the PA to ask for further information if necessary, to diagnose the problem and to repair PA's ontology accordingly. Occasionally, the fault can be tracked down to a particular ontological object (in the worst case, the whole rule), but an appropriate fix cannot be found (we know there is a mismatch but do not have enough information to determine exactly what it is). This ontological object is then marked as unusable.

7. PA uses its updated ontology to replan from its current state (so earlier successful plan steps will not be repeated) and the process begins again. This time, it is hoped that the problematic interaction will be more successful, though it may, of course, fail again due to other mismatches, which in their turn must be identified and diagnosed.

**FAULT REPAIR AND DIAGNOSIS**

The diagnostic process works through a series of algorithms. The flow chart in Figure 1 describes one of these algorithms; this particular flow chart is invoked when the diagnostic process has already determined that a precondition of a rule is at fault. This algorithm determines exactly how it is at fault – is it incorrectly named? is one of the arguments wrong? is the number of arguments wrong? When the diagnostic process has narrowed down what kind of mismatch it is, it next needs to determine the details of the mismatch. Predicate, propositional and domain refinement mentioned in Figure 1 all refer to types of mismatches outlined above; the Shapiro algorithm⁶ is used when there is no structural mismatch but an incorrectly instantiated fact. It determines how this incorrect fact came to be believed. Further details of these processes and the other diagnostic algorithms can be found in (McNeill and Bundy, 2007).

---

⁶ This is so named as it is loosely inspired by Shapiro's procedure for debugging programs by tracing back to find the original source of the problem.
NEGOTIATING REPAIRS

The basic version of ORS makes the assumption that only PA makes repairs: if mismatches were found, it was SA's version which was preferred. In practice, however, this may not always be the best option. It may be that making the proposed change would be very costly to PA – for example, it may require changing an ontological object that is frequently used, and changing it may complicate many other interactions, or the proposed change may involve removing detail that is of high value to PA. PA making the diagnosed repair is not the only way to make the interaction possible: SA making the inverse of the repair would also create an understanding between them that would allow the action to be performed. If, in these situations, it was the case that making the inverse change was not costly to SA then the greatest good would be achieved by SA performing the change rather than PA. We refer to this as *maximising social welfare*.

ORS is a system designed to be used by a particular agent and we do not make any assumptions about the agents with which it interacts, so we cannot assume that they also have access to ORS. If they don’t then they cannot implement repairs themselves and negotiation is impossible: PA must either perform the repair itself or decline the interaction. However, in cases where SA does have access to ORS, negotiation is possible.

In order to reflect the idea that there is a greater cost associated with changing some parts of the ontology than others, we have introduced the notion of protection. An ontological object that is marked as protected will invoke a higher cost if it needs to be altered than one not. There are two levels of protection - high and low. Protection may be applied to an entire ontological object (e.g., a predicate definition or a rule) or to a part of it (e.g., a particular argument of a predicate or a particular precondition in a rule).

With this extended version of ORS, if PA diagnoses a repair that would lead to the alteration of any protected object, it attempts to negotiate with SA to see if SA would implement the inverse repair instead. Because PA is already providing the mismatch information, SA does not need have access to any diagnostic ability, merely the ability to repair its ontology appropriately. In an open environment, SAs will make such decisions on whatever basis they chose. For the purposes of our implementation, we designed our own SAs which would be prepared to perform the mismatches if they had no relevant protection, and would refuse if they had relevant protection at any level. If SA refuses to perform the inverse repair, PA will make the repair itself if it has only low protection; if it has high protection, it will mark SA as inappropriate and try to re-plan.

We have developed a negotiation protocol through which PA and SA can discuss possible allocations of repairs. The negotiation protocol specifies the set of permissible offers and possible agreements. In addition, it provides the agents with the essential vocabulary and the rules for a successful negotiation. The negotiation outcome - which agent will perform the repair - is always the result of mutual agreement. The strategy that each agent follows
is based on a utility-based model: each agent forms its utility for the negotiated repair, and proceeds to negotiate according to this utility. In this the way the agents express their preferences. We have also introduced the notion of an expert agent, where if PA and SA are not sure which of their versions is preferable, they can get the advice of an impartial agent. An expert agent can be any agent (other than PA or SA), nominated by either agent, and will be relevant to the particular repair. For example, if the concern is about a predicate representing money, a banking agent may be consulted.

The lifecycle of negotiation is as follows.

- PA calculates a cost for the repair in the range [0 2]. A cost of zero indicates that no protection is violated, a cost of 1 indicates that low protection is violated and a cost of 2 indicates that high protection is violated.
- PA then calculates the utility of repair, which is the cost of the action not being performed less the cost of performing it (currently this is set at 2 for all actions).
- If the cost of repair is higher than zero, PA initiates the negotiation process, sending a suggested repair to SA.
- Once SA receives this, it calculates its own cost and utility scores for implementing this repair. In our current implementation, SAs calculate this score in the same manner as PA; in an open environment, they can choose their own method of calculating it, providing the utility score is in [0 2]. If its utility score is zero, it has no incentive to perform the repair and will send a rejection notice to PA. If its utility score is greater than zero, it will enter into negotiation with PA, sending back a proposal for interaction, which contains this utility score.
- PA then calculates the social welfare score for either agent performing the repair, where the social welfare of an agreement is a measure of the utility this agreement has for the system in total.
- If the social welfare is maximised by PA performing the repair, it will do so. If the social welfare is maximised by SA performing the repair, PA will request SA to do so. If the social welfare is equal either way then it is not clear which of the agents should perform the repair (SA can agree or decline). In this situation, the agents may chose to consult an expert agent, which will indicate a preference. It is acceptable for either agent to make the repair. However, since the benefits of the different representations is not clear, we introduce the notion of the expert agent to assist in such situations. Both agents have the choice as to whether to accept this recommendation; neither is obliged to recognise the expert agent as an appropriate authority. If both agents do agree to take the expert agents opinion into account, the utility of the agent with the preferred version is increased by 1, and the negotiation continues on that basis.

In order to make our approach flexible, and applicable to agents in an open environment that have not been designed with this specific interaction in mind, we created an ontology which defines the negotiation protocol, so that this is available to all agents without it being hard-coded into the agent.

**EVALUATION**

The standard approach to evaluation of ontology matching, outlined in the OAEI7 is to input two full ontologies and evaluate how many of the mismatches between them are correctly or incorrectly diagnosed, or are missed. Such evaluation makes no sense for

---

7 http://oaei.ontologymatching.org/
ORS, since ORS is designed not to discover and patch all mismatches but only those that are impeding interaction. A better metric for ORS, therefore, is to evaluate how frequently it can facilitate interaction that would have failed without access to its functionality.

We examined mismatches between several large ontologies and analysed which of these mismatches were described by mismatches ORS could diagnose and refine, and which were not. The pie chart below illustrates the results. Although this highlights that there is much ORS cannot currently do, we believe these results to be encouraging. This is the first step towards a new approach to ontology mismatch and ORS is a prototype system, so the fact that it can already tackle 38.8% of mismatches bodes well.

**Figure 2.** Piechart illustrating the percentage of mismatches that fall into the following categories:

1. ORS could refine the mismatch
2. ORS could not refine the mismatch but straightforward improvements to ORS would allow it to solve it.
3. ORS could not refine the mismatch because:
   a. ORS did not have sufficient functionality;
   b. This mismatch is outside the current scope of ORS;
   c. This mismatch is irrelevant to an automated system – this is usually a change to commenting or formatting;
   d. This mismatch could not occur in the representation ORS currently deals with;
   e. This mismatch could not be highlighted in a planning context.
4. The information we had about the mismatch was insufficient to determine which category it would fall into.

Evaluation of the negotiation extension demonstrated that the agents always acted so as to maximise our definition of maximal social welfare. In addition, we used computational tree logic to prove that: deadlock is never reached; when a call for a proposal is received, the receiving agent can always refuse this call or make a proposal; when a proposal is received, the receiving agent can always either refuse it or accept it.

Full details of our evaluation can be found on the ORS website.

**CONCLUSIONS**

In this article, we have introduced ORS, a new approach to ontology mismatch which aims to resolve miscommunication between agents, where this occurs due to ontology mismatches. The differences between these categories and the process by which we determine how to categorise mismatches are not obvious at first glance. Full details of how this categorisation is done can be found on the ORS website (above).

---

8 The differences between these categories and the process by which we determine how to categorise mismatches are not obvious at first glance. Full details of how this categorisation is done can be found on the ORS website (above).

mismatch. ORS works by diagnosing mismatches and repairing ontologies during runtime, but only where this is demonstrated to be necessary. ORS is designed to be a tool that an agent interacting in an uncertain world can rely on to assist it when communication breaks down due to misunderstandings.

We have described the kinds of mismatches ORS can diagnose and refine and have briefly outlined promising evaluation results. We have many ideas for increasing the functionality of ORS which should greatly improve these evaluation results, including broadening the scope of the kinds of ontologies ORS can deal with and building in improved semantic matching techniques by incorporating existing semantic matches. Further details of all aspects of ORS, together with full information about our plans for future work, can be found in (McNeill and Bundy, 2007).

REFERENCES


KEY TERMS & DEFINITIONS

Ontology: formal representation of domain knowledge, containing a class hierarchy and possibly relations and functions between these classes.

Planning agent (PA): an agent which responds to goals through forming plans to achieve them and then enacting these plans through interaction within a domain.

Service Providing Agent (SA): an agent which is able to provide a service to a planning agent, providing the correct conditions are met.
ORS: ontology repair system which equips a planning agent to proceed successfully with problematic interaction by identifying and repairing any ontological problems which may have led to the difficulties.

Structured ontological term: an ontological term, such as a relation or function, the meaning of which is determined not only by the meanings of the words in the term but also by their structure.

Semantic mismatch: mismatch between two ontological terms where the structure is the same but the meanings of the words within the terms is not.

Structural mismatch: mismatch between two ontological terms where the meanings of the words within the terms is the same but their structure is not (two ontological objects may be mismatched both semantically and structurally).

Surprising questions: when two agents are communicating with a particular goal in mind, a question that is posed by one agent which is not thought to be pertinent to the situation by the other agent is considered by that agent to be a surprising question.