

# Facilitating virtual interaction through flexible representation

Fiona McNeill and Alan Bundy

School of Informatics, University of Edinburgh, UK  
{f.j.mcneill;a.bundy}@ed.ac.uk

## ABSTRACT

Fluent, effortless and diverse e-business transactions depend on the ability of systems to interact automatically. The difficulties of tailoring representation and information to be consistent and therefore interoperable needs to fall not on human users but on automated systems. In this paper, we present our system, ORS, which is designed to be a tool for automated agents acting on behalf of people or systems which need to interact, to enable them to understand one another, despite the fact that they are not centrally or consistently designed.

## 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, wide-spread and not under centralised control, such as in the Semantic Web. The simplest solution to this problem is the enforcement of a single ontology: a single view of the world. However, even if all users subscribe to a single ontology, integration problems still exist, as changes and updates are made and users tweak the ontology to fit their own needs. If the agents interacting are from different organisations or fields, even attempting to use a single ontology is usually neither practical nor desirable. Users need to develop a representation that is best for their own data and they need to maintain and update that data locally. The problem of ontology matching has been widely studied and powerful solutions are available (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, 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. Much less considered is the problem of relating compound terms: for example, first-order terms or database entries with multiple fields. In such situations we still have the problem of relating the single terms contained within these compound terms but we must also consider the overall relation of the compound terms, which requires not only semantic but also structural matching. Additionally, traditional ontology matching assumes full knowledge of all relevant ontologies and is performed off-line, prior to interaction: assumptions that are not always practical or valid in large, fast-moving communities, or situations where some information may be confidential. Our hypothesis is that representation – as well as vocabulary and beliefs - must be treated as a fluent and that matching techniques that can map between structured terms are necessary for full integration of disparate data (Bundy et al, 2006). It must be possible to do this matching on-the-fly, as the particular need becomes apparent.

## FOCUS OF APPROACH

Rather than producing a system that is capable of solving these integration problems for all interacting agents – an approach which we believe to be extremely problematic, if not impossible – our approach focusses on equipping a single agent to deal with a dynamic and unpredictable world. If all agents in this world are making use of our techniques, then the whole system of agents is able to deal with this world; however, the work of adaptation is always done by individual agents rather than by a ‘god’s view’ system which can understand and incorporate all participants. In a world in which agents are private and evolving and in which there are a potentially huge number of agents, such a system cannot exist.

In our system, ORS (the Ontology Refinement System)<sup>1</sup>, this individual agent on which we focus is a planning agent (PA). For example, it may be an agent which provides a particular service, such as a selling tickets, and its plan might be to sell tickets to any agents which requested them, provided particular conditions (such as receiving payment) were met. The plans it forms will often involve communicating with other agents or waiting for other agents to communicate with it: for example, waiting for a ticket request to come in, or requesting information from another agent (A1). If communication between PA and A1 breaks down due to some misunderstanding, PA can utilise ORS to analyse the communication thus far and, if necessary, suggest further queries to be made of A1 so that the exact source this misunderstanding can be pinned down and hence corrected. ORS does not take account of other possible sources of communication failure such as agents going off-line: we are solely interested in communication breakdown due to semantic or syntactic misunderstandings. Once the source of misunderstanding has been discovered and patched, PA can make a new attempt to communicate with A1, and this time there is more chance of success (of course, another source of misunderstanding may then be discovered, and this in turn will be analysed and corrected).

Note that this diagnosis and refinement process is triggered only by communication breakdown and its goal is only to resolve the particular cause of the breakdown. A large number of other misunderstandings may exist between two agents and if these are not pertinent to the current communication, they are ignored. This is because ORS is designed to operate in an environment, such as the Semantic Web, where interactions may be frequent and fleeting and where there is no certainty of interacting many times with one agent: interacting with many different agents is often more likely. In such cases, ORS must operate on-the-fly during interaction, since we usually do not know in advance with what we will interact, or precisely what such a communication will involve, and it therefore needs to be lightweight enough to do this without holding up communication. Not only would attempting alignment beyond the needs of the current interaction usually be impossible within such a narrow time frame, it would also very often be wasted, as PA cannot know whether we will interact again with the same agent.

## ONTOLOGICAL MISMATCH

We consider two main types of ontological mismatch:

1. purely semantic mismatch, where the mismatch is between words or phrases – for example, *car* is matched to *auto*.
2. structural mismatch, where the mismatch is between structured terms – for example, *car(Make,Model)* is matched to *car(Make)*<sup>2</sup>.

(Note that we use an initial upper case letter to indicate a variable. As a shorthand, we use the

---

<sup>1</sup> <http://dream.inf.ed.ac.uk/projects/dor/>

<sup>2</sup> Note that we use an initial upper case letter to indicate a variable. In theory, this could be labelled by anything, e.g., *X*. For convenience’s sake, however, we label variables with a term denoting their expected type: for example, *car(Make)* indicates that the argument can be instantiated by any individual of type *Make*.

name of the required type of the argument as the variable label – *i.e.*,  $car(Make)$  is shorthand for  $car(X)$ ,  $type(X,make)$ .)

In addition, for a planning agent there is another kind of mismatch:

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,

*BuyTicket Action: wants\_to\_travel(Me, Destination) -> has\_ticket(Me, Destination)*

matched to

*BuyTicket Action: wants\_to\_travel(Me, Destination) and has\_money(Me) -> has\_ticket(Me, Destination)*<sup>3</sup>.

Of these, the first point is only considered incidentally; the emphasis is strongly on the second and third points. There is already a large body of work aimed at the first point (see (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 example), although often this work is applied in a domain where complete knowledge can be assumed and full matching is desired, rather than the incremental matching of our approach. However, we can integrate this existing work into our structural matching techniques in order to address this problem; therefore, our own research is focussed on the other points.

## Structured Matching

The problem of structural mismatch, is very rarely addressed. However, we believe it to be 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.

In our work, we consider quantifier-free first-order terms; that is, predicates with some number of arguments  $\geq 0$ . Most common service invocations, for example, those expressed in BPEL (\*\*\*) put ref in here), 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 these first-order terms and another more general term is described by the theory of abstraction [cite fausto and toby]. They describe four kinds of mismatch:

- A term is matched to one with fewer arguments – for example,  $car(Make, Model)$  maps to  $car(Make)$ .
- A term is matched to one with a more general predicate – for example,  $car(Make)$  maps to  $vehicle(Make)$ .
- A term is matched to one with a more general type of argument – for example,  $car(Make, Second-hand-dealer)$  maps to  $car(Make, Dealer)$ .
- A term is removed from a rule – for example,  $has(money, Me) -> owns(car(Make), Me)$  maps to  $has(money, Me)$  and  $has(id, Me) -> owns(car(Make), Me)$ .

By inverting these relationships, we obtain four equivalent refinement operators. These abstraction and refinement operations are sufficient for describing most ways in which quantified-free first-order terms can be related but non-identical: terms must be either synonymous, more general, less general or unrelated, and the case where they are synonymous is dealt with by semantic matching. Adding a few additional operators gives us a description of how two unquantified first-order terms may be related.

---

<sup>3</sup> 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.

## WORKED EXAMPLE

Consider the interaction between a travel service agent (TS), which wishes to buy a ticket, and an agent acting as the front-end to a ticket-selling service: we call this agent TB. TB contacts TS with the following message:

TB: *buy(tb,london,edinburgh)*,

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

TS: *money(tb,Amount,dollars)?*

This indicates that TS wishes TB to find a suitable instantiation of the variable *Amount* such that the predicate becomes true. But imagine that TB has money represented as a predicate *money(Agent,Amount)*, and can instantiate this to *money(tb,100)*. Perhaps this agent has operated purely within a single currency zone, and so its designer considered explicit representation of currency unnecessary. TB must therefore respond negatively to TS's enquiry:

TB: *no*

TS: *fail: buy(tb,london,edinburgh)*

On hearing that TB does not have any dollars, TS will refuse to perform the service.

However, TB is able to analyse the problem and alter its ontology appropriately. In this instance, it is obvious that the problem is that *money/3* is mismatched to *money/2*. TB can determine that the first and second arguments of TS's *money/3* match the two arguments of its *money/2*, and that therefore the missing argument is the third one, instantiated by *dollars*. If TB already knows that *dollars* is of type *currency* then it can make the appropriate refinement. If it does not recognise *dollars*, TB must question TS further:

TB: *type(dollars,X)?*

TS: *type(dollars,currency)*.

TB can then add a new type *dollars*, a subtype of *currency*, to its ontology. Once this is done, it can correctly alter its definition of *money* from *money(Agent,Amount)* to *money(Agent,Amount,Currency)*. However, because it is adding information into its ontology, it will not know what this new information will be. How should *money(tb,100,Currency)* be instantiated? It is possible to use heuristics here – for example, if the currency *dollars* has been used in the query, then *dollars* is the most likely instantiation. However, such reasoning is fallible and may lead to errors. A necessarily correct answer cannot often only be obtained through interaction with the user. If TB consults its user and determines that this should, in fact, be *dollars*, it can alter its ontology accordingly and resume interaction with TS:

TB: *buy(tb,london,edinburgh)*

TS: *money(tb,Amount,dollars)?*

TB: *money(tb,100,dollars)*

TS: *success: buy(tb,london,edinburgh)*

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

## THE ONTOLOGY REFINEMENT SYSTEM

The architecture of ORS is illustrated in Figure 1. The flow of control is as follows:

0. The PA (planning agent) starts off with its first-order ontology<sup>4</sup> and a goal that it is attempting to achieve.
1. It first sends its ontology and goal to the translation process, where it is translated into PDDL<sup>5</sup>, an appropriate representation for planning.
2. The PDDL version of the ontology, including the goal, is sent to the planner and a goal is returned.
3. The PA communicates with other agents in order to achieve its goal. After each action is successfully performed, the PA updates its ontology to reflect the changing facts effected by the action.  
If the PA achieves its goal by communicating successfully with every agent it needs to interact with, the plan is finished and the main functionality of ORS is not required. However, if communication fails at any point, the PA performs the steps below.
4. The PA sends information about the point of failure, the agent with which communication was taking place, and any *surprising questions* to the diagnostic system (which also has access to the first-order ontology). These surprising questions are important in the diagnostic process. The PA has preconditions on each of its actions and would normally expect to be asked about some of these before the action could be performed: for example, if the PA is required to have money, the agent performing the action will ask the PA about this money. However, the PA would not expect to be asked about anything outwith these preconditions, since it would not consider these to be pertinent to the action. Any such question would be classified as *surprising*. Note that surprising questions may differ only very slightly from expected questions: for example, the question about money in the worked example would be considered surprising, even though a question about money was expected, because the format of the question is not exactly as expected. Anything that does not *exactly* match a precondition is classified as surprising. Surprising questions are noted but ignored until communication failure occurs: they are then used as the basis for diagnosis.
5. Diagnosis may require further communication with the other agent to determine more about the source of the mismatch.
6. The diagnostic system determines the source of the problem (this process is described in more detail in the following section) and returns this to the PA. The PA sends this information to the refinement system, which alters the first-order ontology accordingly. Note that the PDDL ontology is never updated: this is used for planning and then discarded.

The PA now has an updated ontology but has still failed to achieve the goal. Therefore, it translates its new ontology to PDDL and uses this to replan. The process then begins again, and continues until the PA achieves its goal or until the goal is no longer achievable from its updated ontology.

---

<sup>4</sup> We use ontologies written in KIF (Genesereth and Fikes, 1992), a widely used first-order ontology representation.

<sup>5</sup> The Planning Domain Description Language (PDDL) is the most widely used planning representation and many efficient modern planners use it (Fox and Long, 2003).

## FAULT REPAIR AND DIAGNOSIS

The diagnostic process works through a series of algorithms. The flow chart below describes one of these algorithms; this particular flow chart is invoked when the diagnostic process has already determined that a particular precondition 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? – and when this is determined, refers it to another algorithm to determine exactly how it is wrong. (Note that predicate, propositional and domain refinement all refer to types of mismatches outlined above; the Shapiro algorithm<sup>6</sup> is used when there is no structural mismatch but an incorrectly instantiated fact, and determines how this incorrect fact came to be believed. Further details of these processes and the other diagnostic algorithms can be found in [journal paper].

Note that due to the fact the ORS is working in a domain of uncertainty and incomplete information, the outcome of this process is unknown. In many situations, ORS can give a correct diagnosis which will lead to a correct repair of the ontology. In some situations, ORS can give a correct diagnosis of the problem but there is insufficient information to determine exactly how the ontology should be repaired on this basis. In such situations, ORS marks the particular ontological object as incorrect, but cannot correct it. In some situations, ORS cannot determine the exact cause of the mismatch. In such situation, ORS marks the action rule that lead to the communication failure as incorrect, but it cannot determine how it is incorrect or what other errors in the ontology this may imply.

## EVALUATION

The standard approach to evaluation of ontology matching, outlined in the OAEIO<sup>7</sup>, 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 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 which would have failed without access to its functionality. To do this, we had to add planning context – i.e., action rules and individuals – to ontologies, which usually just contain type hierarchies, functions and relations. This enables them to be used as the basis for interaction in the way required for ORS. Since this is a time consuming process, we did this only for some of the mismatches we found in ontologies, ensuring that a complete cover of all ORS's functionality was obtained, thereby ensuring that ORS was successful in all that it claimed to achieve. In order to determine the value of these mismatches, 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 X% of mismatches bodes well. Full details of our evaluation can be found on the ORS website<sup>8</sup>.

## CONCLUSIONS

---

<sup>6</sup> 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.

<sup>7</sup> <http://oaei.ontologymatching.org/>

<sup>8</sup> <http://dream.inf.ed.ac.uk/projects/dor/>

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 mismatch, by diagnosing mismatches and refining ontologies accordingly during runtime, and 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

- Alan Bundy, Fiona McNeill, and Christopher Walton (2006). On repairing reasoning reversals via representational refinements. In *Proceedings of FLAIRS 2006 (The Florida AI Research Society Conference)*, pages 3–12. AAAI Press, May 2006.
- Bergamaschi, S., Castano, S., and Vincini, M. (1999) Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1).
- 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.
- Fox, M. and Long, D. (2003). PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research*, 20:61–124.
- Genesereth, M. R. and Fikes, R. E. (1992). *Knowledge Interchange Format, Version 3.0 Reference Manual*. Technical Report Logic-92-1, Stanford, CA, USA.
- Gligorov, R., Aleksovski, Z., ten Kate, W., and van Harmelen, F. (2007) Using google distance to weight approximate ontology matches. In *Proceedings of WWW*.
- Kalfoglou, Y., and Schorlemmer, M. (2003) IF-Map: an ontology mapping method based on information flow theory. *Journal on Data Semantics*, 1.
- Fiona McNeill and Alan Bundy (2007). Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *IJSWIS (International Journal on Semantic Web and Information Systems) special issue on Ontology Matching*, 3:1–35, 2007.
- Noy, N., and Musen, M. (2003) The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6).
- 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*.