Diagnosing and Repairing Ontological Mismatches

Fiona McNeill, Alan Bundy, Chris Walton
Centre for Intelligent Systems and their Applications,
School of Informatics, University of Edinburgh
f.j.mcneill@ed.ac.uk,{bundy,cdw}@inf.ed.ac.uk

Abstract. The development of the semantic web ensures that the facilitation of agent communication is an issue of increasing importance. It is usually assumed that agents are using the same ontology and hence can understand one another, but the dynamic and distributed nature of the semantic web mean that this is not always a valid assumption. In this paper, we describe a system under construction that can dynamically discover ontological mismatches between agents during communication and then refine them, so that communication between these agents is facilitated.

1 Introduction

Ontological mismatches between agents can be a serious problem in agent communication, particularly where agents come from different sources, different users or are adapted to different tasks. Even where agents have originated from a single ontology, updates and alterations may make communication impossible. At the same time, since it is advantageous that agents are adapted to their particular tasks, forcing all agents to use identical ontologies is not a sensible or practical solution, nor are there any established mechanisms to enforce this. We are developing a system whereby agents can identify the specific ontological causes of communication failure and adapt these parts of their ontologies dynamically during interactions so that communication becomes possible.

A key concept of the semantic web is the automated processing of ontological knowledge by agents and services. [1] outlines how these are central to the role of the semantic web, and how agents require ontological matching to perform appropriately. However, the nature of the semantic web means that strict controls on these ontologies are impractical, since the semantic web must allow for, amongst other things, partial information, erroneous information and the evolution of ontologies [6]. All of these requirements cause difficulties for agent communication. Our approach is a way of dealing with these ontological mismatches without enforcing tight ontological controls on agents. It is intended to automatically solve a subset of the many problems surrounding ontology mapping, merging and alignment, which are all essential for communication in multi-agent systems. It allows agents to learn about the area in which they are currently active, and thus be more capable of acting successfully within it, without receiving any extraneous information. This is particularly useful in complex and dynamic domains, where a complete and up-to-date representation of the entire domain may be intractable or undesirable.

We are currently working in a planning environment, with an agent forming plans for how to achieve a goal based on its understanding of the domain, and then attempting to execute these plans through communication with other agents. Planning in these complex and dynamic environments is very difficult because any incomplete, incorrect or out of date

information can cause an inexecutable plan to be developed. However, by making productive use of failure, these cases of plan execution failure can be considered to be opportunities to learn more about the domain. Information about the cause of failure is extracted from observation of how and why the plan failed, together with further communication with the other agents involved. Once the point of failure has been located, refinement techniques are implemented to fix the problem, and a new plan is developed using the updated ontology. This plan, though not guaranteed to succeed, is more likely to succeed than the previous plan. This procedure is continued until the goal is successfully reached.

We use *ontology* to refer to the whole of this domain knowledge. The ontology consists of the *signature*, which describes what predicates exist, what arity and type predicates have, the type hierarchy and so on, and the *theory*, which contains the instantiation of the signature for the particular case. We are primarily interested in errors in the signature: for example, flaws in the type hierarchy; incorrect naming of predicates; incorrect arity or type of predicates. However, alterations in the signature will normally also entail alterations in the theory, where the particular signature object is instantiated.

We consider that there are three essential elements to creating such a dynamic ontology refinement system: the ability to extract relevant information about the underlying ontology from the plan; the ability to locate the exact source of the problem; the ability to select and apply appropriate techniques for altering the ontology. These abilities are implemented in different sub-systems within the dynamic ontology refinement system [3]: the *planning system*, which forms and interprets plans; the *diagnostic system*, which locates the source of the problem; and the *refinement system*, which implements the necessary changes to the ontology. In addition, we also have an agent communication system, which creates an environment in which plans can be executed and the necessary communication carried out. The architecture of this dynamic refinement system, and the connections between the sub-systems, is illustrated in Figure 1.

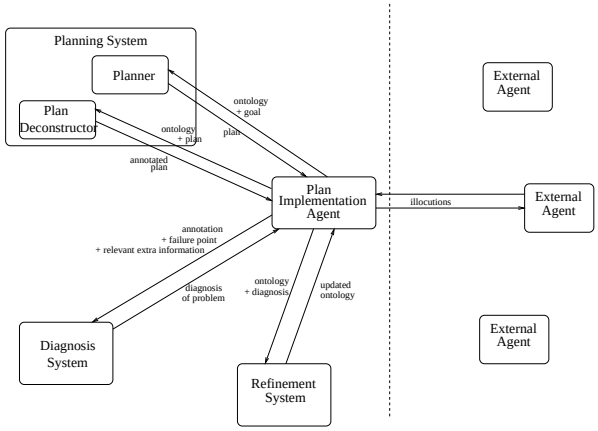

Figure 1: Architecture and interaction of a dynamic ontology refinement system

Such a system is inherently limited: it is very difficult or perhaps impossible to define every possible kind ontological mismatch, and additionally it may be impossible to diagnose differences that are chaotic in nature; for example, names that are altered arbitrarily. Our system relies on there being some method to the changes made: for example, an argument altered to a subtype of that argument. Additionally, we have made many assumptions in order to sim-

plify the problem to the extent where we can produce a working system. The system could later be made more sophisticated to eradicate these assumptions, but in the mean time they reduce the problem to a solvable subset. These include many simplifying assumptions about agent behaviour. Additionally, these refinement techniques are intended for agents which are using similar ontologies that only differ slightly; for example, one agent has an ontology that has been updated or altered, or perhaps one agent is using an old version of an ontology. They are not intended for facilitating communication between agents with completely different ontologies.

Considerable work has been done in the related field of Ontology Mapping, surveyed by Kalfoglou and Schorlemmer [5]. This subject is explored in relation to Semantic Web Services in [2] and [9]. The main differences between such approaches and our work is that we are not assuming that we have full access to both ontologies, but rather that we must glean what information we can about ontologies of other agents from the communication we have with them and from their replies to our queries. Additionally, much of this work is concerned with simply mapping predicates, whereas we also employ more radical techniques such as splitting one predicate into two subpredicates, adding extra arguments to predicates, and so on.

2 Plan Deconstruction

The first issue we consider is that of extracting useful information from the plan. When a plan fails, the only immediate information we have about that failure is the plan step where failure occurs. In order to link this failure to ontological mismatches, we need to have information about which parts of the ontology were referred to in the formation of this plan step. This kind of plan development information is difficult to obtain from planners: the methods employed by most efficient, modern planners do not provide the kind of first-order relational information we need, and planners that are good at providing such information, such as Prolog or Situation Calculus planners, are rather inefficient and thus not desirable as a base for our system. We have therefore developed a *plan deconstructor*, which will take a completed plan from a modern planner, such as Metric-FF, together with the ontology from which the plan was developed, and meta-interpret it to produce an annotated plan [7]. This annotation consists of ontological information justifying each plan step: the rule which describes how this action is performed, and an explanation for each precondition as to why it is believed to be true in the current state. The meta-interpreter steps through the plan, finding this information for each plan step. Thus we have all the semantic advantages of using a first-order planner, without the associated search problems. If plan failure occurs, this annotation is used, in conjunction with further agent communication, to identify the problem.

3 Diagnosis and Refinement

The plan annotation provides a good starting point for discovering the cause of the problem, but, as previously mentioned, is not sufficient. Instead, we need to use the annotation, together with information gleaned during the execution of the plan so far, and details given by the other agents, to diagnose the precise problem. Much information is gained by observing how the agent interaction was proceeding at the point at which the plan failed. For example, did the problem occur immediately after we requested the other agent to perform the task for us or after we replied to a question put to us by the agent? Were there any questions put to us by the

other agent that seemed surprising? By surprising we mean something not precisely contained in our preconditions for that action, thereby indicating that the other agent has preconditions that differ from ours. There are many ways in which a question can be surprising. The name of the predicate may not match the names of any of our preconditions, indicating either that the other agent has additional preconditions or that it has a different naming system to us. It can also be a precondition that we were expecting, but with a different number or different types of arguments. These observations provide vital information about what may have caused plan failure. Additionally, we cannot always expect plans to fail precisely at the problem point in the ontology: perhaps this error has not caused immediate failure, but has skewed the plan process sufficiently that failure will occur at a later stage. In this case we can search back through the plan annotation to find out exactly where failure may have occurred.

3.1 Types of Mismatches

In order to successfully diagnose the kind of failure that has occurred, we need to have a predetermined idea of the kind of failure that might occur. This is an unlimited problem; however, by making the assumption that ontologies will usually be altered in a methodical fashion, we can examine how these changes are normally made to ontologies and thus develop methods for detecting and refining them.

From consideration of the kind of surprising questions that might be formed, we form four types of failure:

- **Case A:** we are faced with an unexpected predicate:
  1. The predicate is related in some way to the name of a predicate we were expecting (e.g. is a subtype).
  2. This predicate is not related to anything we were expecting - either the other agent has an extra precondition or it has a completely different predicate to the one we were expecting.
- **Case B:** we are faced with a predicate we were expecting, but with unexpected arguments:
  3. There are the wrong number of arguments.
  4. The arguments are of the wrong type.

3.2 Refinement Techniques

In order to determine ways in which we may want to add detail to a theory, we examined the field of *abstraction*, which describes how we may remove detail from a theory, and considered how we might invert these techniques. Walsh and Giunchiglia have done much work on abstraction, and have produced a thorough survey of types of abstractions [4]. These, they discovered, fall naturally into four categories. We discovered that when we inverted these to form *anti-abstractions*, these corresponded closely with the kinds of problems revealed by surprising questions, outlined above. Note that in the examples below the first element of a predicate is the predicate name, the following ones are the arguments, question marks before a name indicate variables and arguments without questions marks are constants.

1. **Predicate anti-abstraction** A single predicate is split into one or more subtype predicates, e.g:*(money ?Amount)* maps onto *(dollars ?Amount), (euros ?Amount), (sterling ?Amount)*, etc.
   This is a specific case of type 1 above.
2. **Precondition anti-abstraction** A precondition is added to a rule, e.g:
   (has money ?Agent) => (has item ?Agent) maps onto
   (and (has money ?Agent) (inStock item shop)) => (has item ?Agent)
   This is a specific case of type 2 above.
3. **Propositional anti-abstraction** The arity of a predicate is increased, e.g.:
   (money ?Amount) maps onto *(money ?Amount Dollars),(money ?Amount Sterling)*.
   This is a specific case of type 3 above.

4. **Domain anti-abstraction** The type of an argument is divided into one or more subtypes, e.g.: *(money ?Amount European)* maps onto *(money ?Amount Euros), (money ?Amount Sterling),(money ?Amount Krona)*
   This is a specific case of type 4 above.

Thus these formal descriptions of ontological differences can provide us with a means of addressing some of the specific problems we expect to encounter, and a basis from which to implement refinement techniques. They can also be applied in reverse in order to remove unwanted detail from an ontology. An example of the kind of algorithms we are using is given below. This is for the situation in which PA has been asked a surprising question. SQP refers to the predicate in the surprising question.

```
if name SPQ = name of a precondition of the action, P
   then
   if   arity SPQ equals arity of P
      then  check the types of the arguments of these predicates:
         domain (anti-)abstraction may be applicable
      else discover the type of the extra/missing argument:
                propositional (anti-)abstraction is applicable
   else
   if  name SPQ is a sub/super-type of a precondition
      then predicate (anti-)abstraction is applicable
      else      precondition (anti-)abstraction may be applicable
```

Many types of signature errors will be immediately apparent: for example, if we encounter a predicate with an extra argument, the problem in the signature is obvious. However, some signature errors will not cause immediate plan failure. Incorrect postconditions in a rule, for example, may cause facts to be believed that should not be believed, or vice versa. Such problems may not cause plan failure, in which case they are not addressed, or they may cause failure at some point later on in the plan execution. If the cause of failure is identified as a instantiated predicate that has no obvious signature errors, then it is necessary to search back through the plan justification for a signature error further back. Our methods of searching back through the plan justification to locate the point of error are inspired by work by Shapiro [8] on algorithmic program debugging.

We need also to deal with failure in a situation where less information is available: perhaps an action fails immediately after a request for it to be performed, without any additional questioning, or perhaps failure occurs despite all the questions being expected and replied to in a way that is considered to be appropriate. We have developed algorithms to apply in both these situations that can diagnose what the problem is. The latter case is the most difficult, since information about the other agent's ontology is limited, and thus in some cases we may not be able to diagnose precisely how this ontology differs from PA's. In such cases we can merely add to the ontology the information that the particular action cannot be performed by the particular agent in the given circumstances, without being able to describe how to solve this problem. Such failures as this are inevitable in situations where we are working with limited information and are still helpful to some degree as they make it clearer what actions cannot be performed.

3.3 Applying the Refinement Techniques

Once we have these diagnostic algorithms in place, we can accurately identify many kinds of ontological mismatch, and thus we can refine the ontology according to the diagnosis. It

is always conceivable, however many of these types of errors that we identify, that we will encounter an ontological mismatch that is different to anything we expect. Such cases are beyond the scope of the current project, where our focus is to describe as many predefined ways in which ontologies can differ as possible and implement these, based on the techniques described by the abstractions and anti-abstractions. There will always be ontological mismatches that will slip through the net of our refinement system.

4 Conclusions and Further Work

We have identified the problem of ontological mismatch as a major factor in agent communication and service failure, particularly in cases where the agents and services are different adaptations of shared sources or have different developers. We have argued that dynamic refinement of the specific causes of failure is the most appropriate method of dealing with this problem, and have outlined some of the methods that we employ to identify and fix these mismatched ontological arguments. Even though the application of our methods does not always guarantee that the subsequently formed plan will be executable, it gives a procedure for learning more about the domain in which they are interacting, and how to communicate with the agents or services that they find there, and thus makes them more capable of interacting more successfully within that domain. Currently, the planning and diagnostic systems are fully implemented. We are now working on applying the diagnosed corrections to the ontology and replanning from this basis.

References

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web, May 2001. Scientific American.

[2] Mark H. Burstein. Ontology mapping for dynamic service invocation on the semantic web. In *Proceedings of Semantic Web Services Symposium*, 2004 AAAI Spring Symposium Series, March 2004.

[3] A. Bundy F. McNeill and M. Schorlemmer. Dynamic ontology refinement. In *Proceedings of ICAPS'03 Workshop on Plan Execution*, Trento, Italy, June 2003.

[4] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 56, 1992.

[5] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18:1:1–31, 2003.

[6] Marja-Riitta Koivunen and Eric Miller. W3c semantic web activity. In *Proceedings of the Semantic Web Kick-Off Seminar in Finland*, November 2001.

[7] Fiona McNeill, Alan Bundy, Chris Walton, and Marco Schorlemmer. Plan execution failure analysis using plan deconstruction, December 2003. http://planning.cis.strath.ac.uk/plansig/index.php ?page=past22.

[8] Ehud Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, 1982.

[9] Nuno Silva and Joao Rocha. Ontology mapping for interoperability in semantic web. 2003.