

# Planning from rich ontologies through translation between representations

Fiona McNeill, Alan Bundy, Chris Walton

Centre for Intelligent Systems and their Applications,  
School of Informatics,  
University of Edinburgh  
{f.j.mcneill,a.bundy,c.d.walton}@ed.ac.uk

## Abstract

The richness and expressivity of standard ontology representations and the limitations on expressivity required by modern planners have resulted in a situation where it is hard for an agent both to have a rich ontology and be capable of efficient planning. We discuss how translation between different kinds of representation can allow an agent to have different versions of the same ontology, so that it can simultaneously meet different demands of expressivity. We introduce our ontology refinement system (ORS), in which these ideas are implemented.

## Using rich ontologies for planning

There is currently a disparity between the richness of ontological representation used in multi-agent systems, and ontologies used in Semantic Web like environments, and the richness of ontological representation used in planning. This disparity comes about due to the different requirements of each domain.

In a multi-agent system and environments such as the Semantic Web, rich, expressive ontologies are desirable. Such ontologies facilitate the encoding of detailed domain information: information about infinite domains, meta-information about ontological objects, complex class hierarchies which allow for slot information in classes, use of the open world assumption, and so on. Some common ontological representations for multi-agent systems, such as Knowledge Interchange Format (KIF) [3], are full first-order, and are thus extremely expressive. Other common ontological representations: for example, description logic based ontologies, such as RDF and OWL, are less expressive than full first-order logic, owing to the tractability problems associated with inference in full first-order logic, but, nevertheless, retain a high level of expressivity.

In popular planning representations, much of this expressivity is removed. Languages such as PDDL [2], resemble first-order languages; however, this is an illusion. Most planners that take domain information from PDDL files are propositional and thus, though

PDDL provides a first-order window on to the propositional space, anything that is expressed in PDDL must be translatable into propositional logic. This places restrictions on what can be expressed; there are some ontological objects that are expressible in a first-order representation but not in a less expressive representation: for example, quantification over infinite domains. Additionally, the closed world assumption is normally used in planning.

One might argue that this disparity comes about partly due to the separation of the planning community and the ontology community: state-of-the-art planners are usually designed and assessed on how well they perform purely with respect to planning considerations; there is much less emphasis on how to balance good plan formation performance with consideration of other issues, such as dealing with richer ontologies. However, there is a more fundamental issue underlying this disparity. Automated planning is very difficult, largely because the search problems involved in finding even short plans are vast. The only feasible way to solve these problems is to reduce the search space. Thus the most important aspect of planning representations is that they are not difficult to search through; this inevitably leads to loss of expressivity.

There are two approaches to this problem. One approach is to attempt to balance the demands of an expressive ontological representation with those of a tractable planning representation. The resulting representation will be less expressive than a standard ontological representation and less efficient for producing plans than a standard planning representation; however, the advantage of combining both facets in a single representation may be thought to outweigh these problems. However, we believe that the best solution to the problem is provided by an alternative approach: allowing ontological knowledge to be represented in different ways, depending on the current required functionality, and translating between the different representations as necessary. Inevitably, information is lost through translation from a more expressive to a less expressive representation. However, if the most expressive representation is retained after it

is translated to a less expressive representation, then the agent still has access to its complete ontology as well as to the less expressive representation that can be used, for example, for planning. We believe that the demands of a planning representation are incompatible with the demands of a standard ontological representation. The ability to form plans quickly and efficiently is vital to agents that are attempting to plan within multi-agent systems, but equally, the ability to represent complex information within their ontology is important. We believe that an attempt to combine the two needs in a single representation requires too great a loss to both domains, and therefore our approach is to develop translation processes between different kinds of ontological representations.

It should be noted that by ontology, we mean both the representation language of the domain and the knowledge base expressed in that language. Thus, in our terms, a PDDL ontology would consist of a domain file and one or more problem files; a KIF ontology would define the vocabulary but also contain the facts expressed in that vocabulary. Therefore, an ontology can be altered either by changing the representational language or by changing the facts expressed in that language (as occurs during plan execution).

In this paper, we describe the translation process that we have implemented, and explain its role in ORS. We describe the context that ORS is designed to work in: that of a multi-agent, service-based architecture, and mention how this context affects the kind of translation that is necessary in ORS. We discuss how our evaluation of ORS demonstrates that this translation process is successful, and show how, as a result, ORS can be used to dynamically refine ontologies in a planning environment.

### Translating from KIF to PDDL

Currently, we have implemented one such translation process: translating from KIF ontologies to a PDDL representation. In translating from KIF, we have already tackled some of the most severe problems inherent in such an approach: KIF is full first-order, and thus the loss of expressivity in our existing translation process is at least as severe as the loss of expressivity in translating any ontological representation to PDDL, although our system does not currently deal with full KIF but only with a subset of it; thus not all these issues have been confronted. Certainly, there would be different implementation issues when translating from a language such as OWL to PDDL, but the theoretical problems surrounding loss of expressivity would be less. Full details of this translation process can be found in [7]; in this section we briefly discuss some of the chief issues involved in this process, which is illustrated in Figure 1.

We have implemented this translation process as part of our ontology refinement system (ORS), which is discussed in the following section. The translation process is important because the system deals with

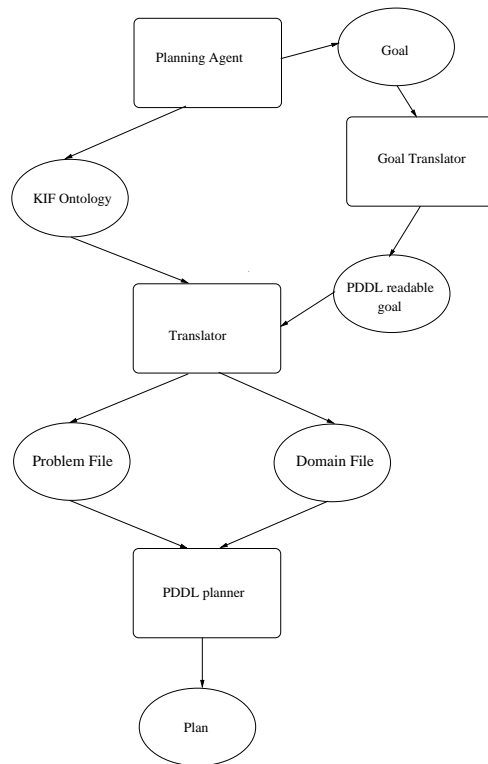


Figure 1: Architecture of Translation System

agents that are operating within a multi-agent system, but which also need to form plans. In our system, the KIF ontology is considered to be the definitive ontology: the agent's true understanding of the state of the world is represented in the KIF ontology. Other representations are used only when this is practically necessary for the agents, and any changes made to these other representations - for example, by actions being performed - must be made to the KIF ontology also, so that the KIF ontology is always up to date with respect to the agent's understanding of the world. In particular, the KIF ontology is translated to PDDL when the agent wishes to form a plan. The difference in expressivity between KIF and PDDL mean that the PDDL representation is not completely equivalent to the KIF ontology. However, the agent has not lost information during this translation process, because it still has access to the original KIF ontology; it is simply that the agent's full ontology is not necessarily represented in PDDL. This loss of expressivity has a disadvantage in that it is possible that there are valid plans that could be formed from the KIF ontology that cannot be found from the more limited PDDL ontology; however, it has a strong advantage in that it is now possible to use this knowledge to efficiently form plans, and, moreover, this loss of information is concerned only with the formation of this particular plan and does not affect any other parts of the system.

In terms of expressivity, there are some ontologi-

cal structures that are expressible in a full first-order representation but not in a representation that must be translatable into propositional logic; for example:

1. quantification over infinite domains,
2. uninstantiated variables

In our system, the first of these issues is not a concern, because it does not currently deal with KIF ontologies that contain quantification over infinite domains. A complete translation process that dealt with such KIF ontologies would need some way to represent this in PDDL, which might be through some kind of finite abstraction. However, this would create larger expressive differences between the original KIF and the resulting PDDL than we have currently experienced.

The second of these issues, however, is a concern. Not allowing uninstantiated variables in the representation constrains a plan to involve only individuals that are already present in the ontology. In many planning situations, that is quite acceptable; it is usually desirable for a plan to be fully instantiated before execution commences. However, there are some situations where this is not the case. Consider, for example, a plan which involved booking a flight, for which a flight reservation number was given, and then automatically checking in for the flight, using this reservation number. The fact that such a reservation number exists is important during plan formation; however, it is not only unnecessary, but impossible, to know the particular instantiation of the number before plan execution begins; this can only be instantiated during execution. In such situations, the propositional restrictions of PDDL present problems. We circumvent this problem through the use of *pseudo-variables*, which are declared as individuals when the PDDL files are produced, but are then uninstantiated by the agent when it interprets the plan produced by the planner. Thus, for example, an action rule that involved buying a ticket would force this reservation number to be the individual PV1. The translation process keeps track of how many pseudo-variables must be declared, which pseudo-variable refers to which uninstantiated object, and so on. Details of how this is done are given in [7].

There are many further implementational difficulties in the translation process. Many of these centre around the fact that PDDL-based planners keep track of the state whilst the plan is being formed, whereas such a concept has no meaning for a static ontology. Thus, for example, the value of numerical functions is automatically tracked in a PDDL planner and does not need to be stated explicitly, as it would in a static ontology. In PDDL, an initial declaration is made (if appropriate); for example:

```
(= (Money ?Agent) 1000)
```

and thereafter `(Money ?Agent)` can be referred to without explicit reference to this value; the planner keeps this information explicitly in its internal knowledge base, and thus this does not need to be represented explicitly in PDDL. If calculations are per-

formed, these are again done without any explicit reference to the value; for example:

```
(> (Money ?Agent) (Price ?Item))
```

means that the value attached to `(Money ?Agent)` must be greater than the value of `(Price ?Item)`.

In a static representation such as KIF, such values have to be declared explicitly because there is no mechanism for keeping track of them, as there is in a planner. Thus instantiated numerical functions are declared in the same way as any other instantiated predicate; for example:

```
(Money ?Agent 1000).
```

The function is always referred to in this way. Calculations such as the one described above must be represented using these explicit arguments; for example, the above calculation would be represented as:

```
(Money ?Agent ?Amount) ^
```

```
(Price ?Item ?Cost) ^
```

```
(> ?Amount ?Cost)
```

Such different requirements force extensive rewriting of the KIF ontological objects, particularly the action rules, in order to create valid PDDL ontological objects.

There are several other issues that need to be dealt with in such a translation, which are detailed in [7]; for example, dealing with arithmetic operators. However, most of these are merely a recoding of information, and do not affect the expressivity.

As mentioned above, our translation does not deal with full KIF, but with a constrained version that is sufficient to express the ontological information required in our system. A full KIF ontology would present further translation issues. We believe, however, that it is possible to produce a PDDL version even of full KIF that is correct, though not complete, with respect to the KIF ontology. This translation process allows us to take an abstraction of a rich ontology, so that it can be used in efficient planning.

The way in which ontologies are handled in our system requires translation only to be one way; we have not done any work on translating from PDDL back to KIF. Creating a valid KIF ontology from the PDDL ontology should not, on the whole, be much harder than translating from KIF to PDDL; for example, instead of folding a numerical predicate `(Money ?Agent ?Amount)` into `(Money ?Agent)`, this would be unfolded by adding a variable name. This variable name could be arbitrarily chosen, but would need to be consistently used: for example,

```
(> (Money ?Agent) (Price ?Item))
```

would first be converted to

```
(> ?Var1 ?Var2)
```

and then the meanings of these variables would need to be declared appropriately:

```
(Money ?Agent ?Var1) ^ (Price ?Item ?Var2)
```

However, since there is usually some loss of expressivity in the translation process from KIF to PDDL, a retranslation of the PDDL ontology would result in a

KIF ontology that was likely to be less expressive than the original ontology. It is thus better, if the situation allows, to always translate from a more expressive ontology to a less. Thus the most expressive version of the ontology can be considered to reflect the true understanding of the agent, and less expressive versions are produced when they are necessary, used to perform their role, such as planning, and then discarded. If any changes produced when using these less expressive ontologies are made directly to the most expressive ontology, rather than completely retranslating the less expressive ontology, then the most expressive ontology retains its full expressiveness, whilst also being kept up to date.

### Example Translation

Consider the situation in which a planning agent (PA) is given a goal to purchase an on-line plane ticket. In order to achieve this goal, several steps must be carried out. For example, the agent must locate a ticket-selling agent, it must ensure it has sufficient funds, it must work out the correct origin and destination for the flight, and so on. Clearly, before the agent can act, it must have a plan for how to achieve the goal. Therefore, as soon as the agent identifies a goal, it sends the whole ontology, together with a suitable representation of this goal, to the translator. PDDL files for the ontology are produced, which can then be sent to the planner. Once the PA has the plan, it can then begin to execute the plan steps. In this short example, we have the following ontological objects in the original KIF ontology:

```
(Define-Frame PA :Own-Slots
  ((Instance-Of Agent)) :Axioms ((Money
    PA 500)))

(Define-Frame Edinburgh :Own-Slots
  ((Instance-Of City)) :Axioms ((Flight
    Edinburgh London 300)))

(Define-Individual London (City) "")

(Define-Function Flight (?Place-0
  ?Place-1) :-> ?Value "" :Def (And
  (Place ?Place-0) (Place ?Place-1)
  (Number ?Value)))

(Define-Function Money (?Agent-0)
  :-> ?Value ""
  :Def (And (Agent ?Agent-0)
    (Number ?Value)))

(Define-Class Agent (?X) ""
  :Def (And (Thing ?X)))

(Define-Class City (?X) ""
  :Def (And (Place ?X)))

(Define-Class Place (?X) ""
  :Def (And (Thing ?X)))

(Define-Axiom Book-Flight "" :=
  (=> (And (Flight ?Agent-Loc ?Conf-Loc
```

```
      ?Price)
      (Money ?Agent ?Amount)
      (< ?Price ?Amount))
  (And (Has-Ticket ?Agent)
    (= ?Newamount (- ?Amount
      ?Price))
    (Money ?Agent ?Newamount)
    (Not (Money ?Agent ?Amount))
  )))
```

There are objects referred to in the axiom that are not defined in the ontology section above: these are omitted for brevity.

Our translation would produce the following PDDL domain file from the above KIF ontology:

```
(define (domain domain Ont)
  (:requirements :strips :fluents :typing)

  (:predicates
    (Agent ?Agent)
    (Place ?Place)
    (City ?City)
  )

  (:functions
    (Money ?Agent)
    (Flight ?Place1 ?Place2)
  )

  (:action Book-Flight
    :parameters (?Agent ?City1 ?City2)
    :preconditions (And
      (< (Flight
        ?City1 ?City2)
        (Money ?Agent)
        (Agent ?Agent)
        (City ?City1)
        (City ?City2))
      :effects (And (Has-Ticket ?Agent)
        (decrease
          (Money ?Agent)
          (Flight ?City1
            ?City2)))
    ))
```

and the following PDDL problem file:

```
(define (problem problemOnt)
  (:domain domainOnt)
  (:objects London Edinburgh PA)
  (:init
    (Agent PA)
    (City London)
    (City Edinburgh)
    (= (Money PA) 500)
    (= (Flight Edinburgh London) 300)
  )
  (:goal
    (Has-Ticket PA)))
```

### Ontology refinement in a planning context

In this section, we briefly introduce our ORS system, to illustrate the role of the translation process in the system, and the role of the two different ontologies.

More detailed information about ORS can be found in [1, 6].

The central function of ORS is to allow agents to refine their ontologies when they discover that they are incompatible with the ontologies of other agents. It is common in current multi-agent systems that an assumption is made that agents have the same ontology; ontological incompatibility leads to failure. However, this is often not a reasonable assumption. Off-the-shelf ontologies are frequently updated, as they are found, during use, to be too limited for the task required, or to be encoding excessive information that is found to be unnecessary, or the ontology moderators decide it would be useful to extend or restrict the domain encoded in the ontology. This results in off-the-shelf ontologies that exist in several different versions. Additionally, individual users might take an off-the-shelf ontology and alter it for their own ends. Thus it is not uncommon for agents to have ontologies that are broadly similar, but that are different in certain respects, and for these differences to result in failure of the agents to interact successfully.

If two agents have vastly different ontologies, it is difficult to see how they could interact in a fully automated manner, since they would have no basis for understanding one another. However, if two agents have ontologies that are, for the most part, the same, but that differ in some respects, then there is potential for fully automated interaction between these agents, even when they need to deal with the parts of their ontologies that are mismatched, because the parts of their ontologies that they share gives them a basis for understanding one another, and terms in which to discuss what their ontological mismatches might be. The purpose of ORS is to allow agents to use this shared portion of their knowledge to diagnose how their ontologies are mismatched, and to patch their ontologies so that successful interaction becomes possible.

The way in which two different versions of an ontology may differ depends on their representation. A common difference may be the removal or addition of complete ontological objects. However, a more interesting difference is when existing ontological objects are modified in some way. In a first-order ontology, such as KIF, this may happen in, for example, the following ways:

- changing the arity of a predicate so that it could encode more or less information;
- changing the name of a predicate: this is most interesting and easier to detect if the name is changed to a related name, perhaps a subclass or superclass of the existing name; for example, the predicate `Money` may be changed to the predicate `Dollars`;
- changing the class requirement for an argument of a predicate: this, again, may involve changing the class to a sub- or super- or otherwise related class; a similar alteration is a change in the order of argu-

ments in a predicate;

- an action rule may be altered by adding or removing a precondition, or by adding, removing or otherwise altering an effect of an action.

Ontological mismatch is not inherently related to planning; this could occur in any situation where ontologies are used by inference. However, we have been investigating this problem in a planning context. ORS is used within a multi-agent system, in which the agents are either planning agents (PAs), or service-providing agents. Although the system is compatible with the existence of more than one PA, we make the assumption that only one plan is being executed at one time, and thus consider that there can only be a single active PA in the system at any one time. This assumption is not compatible with complex multi-agent systems, and future versions of the system will work on relaxing this assumption. In ORS, plan steps are always tasks that can be performed by other agents: for example, a `buy-ticket` action might be performed by a *ticket-selling agent*.

The most significant way in which our system differs from standard methods of ontology mapping and merging [4] is that we do not assume that we have access to all of both ontologies. Instead, we assume that we have access to all of one of the ontologies (the PA's), but that the other ontology, that of the service-providing agent, which is not usually owned by the same user as the PA, is only revealed through direct questions that are put to the service-providing agent by the PA, and by information gleaned from plan execution failure. We believe that this is a more realistic approach in situations such as agent interaction in large multi-agent systems, as not only is it impractical to completely map two ontologies where only a small change may be necessary, but also there may be concerns about secure and commercially sensitive information that mean agents would be unprepared to reveal their entire ontologies. Additionally, we assume that any agent operating in such an environment would be able to answer direct questions and, if it is a service-providing agent, perform tasks; it is not a normal part of agent interaction to reveal large sections of ontology, and we therefore cannot expect it of agents. Two agents may have very different underlying representations but still be able to interact via an agent protocol; thus each agent must glean sufficient information via this protocol; fully mapping the two ontologies is not helpful.

Exploring ontological mismatch in a planning context is facilitated by the fact that a planning context provides a clear indication that some kind of ontological mismatch has occurred: a service-providing agent refuses to perform an action that the PA believed to be performable in the current situation, thus indicating that the service-providing agent and the PA are not using identical ontologies; and a clear indication that the patching performed has removed the particular mismatch: the previous point of failure no longer

causes a problem.

The architecture of ORS is illustrated in Figure 2, and is as follows:

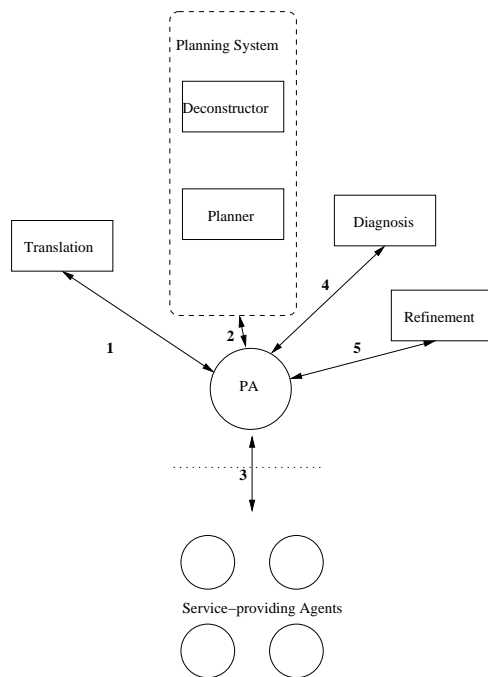


Figure 2: Architecture and interaction of the dynamic ontology refinement system.

1. The PA sends its KIF ontology to the translation system. The trigger for this is the PA receiving a goal that it is required to fulfil. The translation system returns appropriate PDDL files, with the goal correctly inserted, and also returns a version of the ontology in Prolog readable syntax, which is required for direct interpretation of the ontology by the PA, and also in the plan deconstructor (see step 2).
2. The PA sends the PDDL files to the planner, which produces a plan for achieving the goal. This plan is interpreted by the agent and translated into a Prolog readable version, which is then sent, together with the Prolog readable version of the ontology, to a *plan-deconstructor*, whose role it is to link the plan steps to the underlying ontology. One drawback of using propositional planners, which we do not discuss in detail here, is that they cannot provide first-order level information about how the plan produced is related to the underlying ontology: what action-rule was used to perform each action, why the preconditions of these rules were thought to be valid in the particular situations, and so on. This information is sometimes important in linking plan execution failure to mismatches in the underlying ontology. The purpose of the plan deconstructor is to provide this information. The deconstructor

steps through the produced plan, meta-interpreting it with respect to the ontology, and returns this information as a *justification* of the produced plan. It thus acts in a similar way to a first-order planner, but circumvents the massive search problem faced by such planners by using a plan that has already been produced by an efficient planner. Further information about this process can be found in [8].

3. The plan, annotated with a justification for each step, is returned to the PA and execution begins. This execution occurs in an agent communication system, where the PA can locate the agents with which it needs to interact.
4. If failure occurs, information about the communication thus far, together with the relevant parts of the justification, are sent to the diagnosis system. Further agent communication is usually required to pinpoint the exact source of the problem. In some situations, it is impossible to accurately diagnose the source of the mismatch.
5. If an exact, or at least plausible, diagnosis can be made, this diagnosis is passed to the refinement system, which implements the relevant change to the KIF ontology of the agent.
6. The process is repeated, with the updated KIF ontology being retranslated into PDDL, and a new plan formed. This is repeated until the goal is achieved, until the diagnosis system fails to return an applicable diagnosis or until the refinements to the ontology determined by the diagnostic process result in an ontology from which it is not possible to reach the goal.

### Example Mismatches

Before a service-providing agent can perform a service for a planning agent, it must ensure that all the preconditions for performing the service are fulfilled. The value of some of these preconditions it can ascertain for itself, some must be checked with other agents, and some must be checked with the PA. For example, the service-providing agent may need to check how much money PA has, so that it can ensure this is enough for providing the service. It might thus put the following question to PA:

SPA: (Money PA Dollars ?Amount)

By consulting the example ontology in the previous section, we can see that this does not correspond to PA's ontology, where money is represented as a binary predicate and does not include the `Currency` argument. Thus PA cannot appropriately respond to the service-providing agent's question, and must reply:

PA: no.

The service-providing agent will then refuse to perform the service for PA, because it can see that the preconditions are not met. PA will then use the diagnostic system to analyse why failure occurred, which in this case is fairly clear: there is a mismatch

between the service-providing agent's representation of Money and the PA's. The PA must then refine its ontology (described above), so that the following changes are made:

```
(Define-Frame PA :Own-Slots
  ((Instance-Of Agent)) :Axioms ((Money
    PA MetaVar 500)))

(Define-Function Money (?Agent-0
  ?Currency-0)
  :-> ?Value ""
  :Def (And (Agent ?Agent-0)
    (Currency ?Currency-0)
    (Number ?Value)))

(Define-Axiom Book-Flight "" :=
  (=> (And (Flight ?Agent-Loc ?Conf-Loc
    ?Price)
    (Money ?Agent ?Currency
    ?Amount)
    (< ?Price ?Amount))
  (And (Has-Ticket ?Agent)
    (= ?Newamount (- ?Amount
    ?Price))
    (Money ?Agent ?Currency
    ?Newamount)
    (Not (Money ?Agent ?Currency
    ?Amount))
  )))
```

Note that all the ontological objects mentioned in the first example but not listed here are those that are not affected by the change.

Once these refinements have been made, the process is repeated. After translation this time, the ontology will become:

```
(define (domain domain Ont)
  (:requirements :strips :fluents :typing)

  (:predicates
    (Agent ?Agent)
    (Place ?Place)
    (City ?City)
  )

  (:functions
    (Money ?Agent ?Currency)
    (Flight ?Place1 ?Place2)
  )

  (:action Book-Flight
    :parameters (?Agent ?City1 ?City2)
    :preconditions (And
      (< (Flight
        ?City1 ?City2)
        (Money ?Agent
        ?Currency))
      (Agent ?Agent)
      (City ?City1)
      (City ?City2))
    :effects (And (Has-Ticket ?Agent)
      (decrease
```

```
(Money ?Agent
  ?Currency)
(Flight ?City1
  ?City2)))
))
```

and the following PDDL problem file:

```
(define (problem problemOnt)
  (:domain domainOnt)
  (:objects London Edinburgh PA)
  (:init
    (Agent PA)
    (City London)
    (City Edinburgh)
    (= (Money PA Dollars) 500)
    (= (Flight Edinburgh London) 300)
  )
  (:goal
    (Has-Ticket PA)))
```

Note that during refinement of the KIF ontology, the Money fact changes from (Money PA 500) to (Money PA MetaVar 500). This MetaVar is used because we cannot know the correct way of instantiating this new variable. The class of MetaVar is restricted to being Currency because of the function definition of Money. However, when this ontology is used for planning, this variable must be instantiated. An appropriate instantiation is therefore chosen, and the fact becomes (= (Money PA Dollars) 500). This creates some risk of error, as we cannot be sure that Dollars is the correct instantiation.

We may deduce from this refinement that a currency argument is also relevant to modify the Price argument of the flight function: this may become (Flight ?Origin ?Destination ?Price ?Currency)

However, ORS does not make any such deductions, refining only those ontological objects for which it has direct evidence of mismatch. If this refinement were necessary, so would only come to light through further plan failure.

## Evaluation of ORS

The implementation of ORS described above has been completed, and has been evaluated with respect to off-the-shelf ontologies for which we have different versions. We have been somewhat hampered in this endeavour by the fact that is not easy to find existing ontologies for situations such as the ones we are investigating. Because the system is designed primarily for an environment that is still in development, the Semantic Web, there are not many existing ontologies for such situations. Additionally, the planning community tends not to keep large bodies of ontologies that have been updated over time, since the ontologies themselves have not historically been of much interest to the planning community; they are built to provide a basis for testing planners. The continual update and alteration of ontologies that is found in the more traditional ontology communities has not been so impor-

tant in planning. The very problem that the system is attempting to aid: that of closer interlinking between standard ontologies and planning; has made it difficult to find suitable ontologies with which to evaluate the system. The most important requirement for our evaluation, which is the most difficult to fulfil, is that different versions of the ontology should be available, so that we can test our system against mismatches that agents would really encounter if they were using different versions of the same ontology.

Our solution has been to take existing ontologies, designed for encoding complex information but not for planning, and overlay a planning scenario on top of these. Such ontologies are not immediately appropriate for planning, not solely because of the expressivity issues, which ORS is specifically designed to deal with, but also because they are static, and not designed for use in a dynamic situation. There are very few, or no, action rules that tell us how to alter the ontology, as would be found in an ontology designed for planning. We have therefore taken these ontologies and added such information to them, so that we can use them in a planning domain.

We have tested ORS using six different ontologies. Three of these are off-the-shelf ontologies: PSL (Process Specification Language) [10], SUMO (Suggested Upper Merged Ontology) [11] and AKT (Advanced Knowledge Technologies) [9]. The other three are planning ontologies for which we have developed plausible ontological mismatches: a blocks world ontology, a lift scheduling ontology and a conference booking ontology.

We have demonstrated that the system can be successfully used to translate these ontologies, form plans from the translated ontology, execute these plans until an ontological mismatch causes plan execution failure, diagnose the root of the mismatch, patch the original ontology accordingly and retranslate and replan from this updated ontology. Many of the mismatches we have encoded between the PA and the service-providing agents are genuine mismatches that would occur if they were using different versions of these ontologies.

Another aspect of ontologies that makes evaluation of the system difficult is that ontologies are currently updated on the assumption that these updates will be read and interpreted by humans. Thus, although many of these changes can be detected automatically by our system, many cannot. For example, there is little attempt to describe new ontological objects directly in terms of existing ontological objects so that an agent can interpret how they should fit in to its ontology; instead, devices such as using similar names are used, and commenting is used to describe what has been done, so that it is immediately obvious to a human user how these new objects fit into the ontology, but it is difficult to deduce this in a fully automated manner. We believe that if systems such as ORS become more widely used, more effort will be made to update on-

tologies in a way that would facilitate the automated patching of mismatches, and thus the process of ontology refinement will become easier.

## Conclusions

The integration of different fields of AI is extremely important to the development of the subject. The techniques developed in planning and scheduling could be very useful to many other fields, such as the Semantic Web, e-commerce, multi-agent systems, and so on. The application of planning to these domains is hampered by the different representational requirements. We believe that forcing representational constraints on users to make these fields more compatible will not be successful: ontologist will not be willing to lose the current richness of ontologies; planners will not be willing to make do with less efficient planners. We suggest, therefore, that the most appropriate solution to this problem is to allow many different representations of the same, or similar, knowledge to exist simultaneously, and that work must be done on developing translation processes between these representations.

In this paper, we have described how we have done this for two representations: KIF and PDDL; and how this has allowed us to develop a working system where agents have extremely expressive ontologies, but are also capable of efficient planning. We have described our ORS, which is currently fully implemented and can successfully refine ontologies, both in plausible planning situations and with genuine ontological mismatches gleaned from off-the-shelf ontologies that are available in different versions. An important aspect of this system is the ability to handle and translate between different ontological representations, so that an agent can use different levels of expressivity, depending on the task at hand.

This approach could be extended to many different ontological representations. A translator between DAML and PDDL has already been developed [5]. There are different implementation issues in the different translation processes, and different degrees of expressivity loss, but there is no reason why a correct, though not necessarily complete, version of any ontological representation cannot be rendered in any other. As well as creating further translation processes, we are also interested in using these with ORS to allow ontology refinement for different ontological representations. The potential mismatches that would occur would vary: a representation such as OWL could not be altered in the same way as a full first-order representation, and thus some effort would be required to adapt ORS to each representation. Nevertheless, the framework provided by the system is not representation dependent, and could be used in many different circumstances.

## References

- [1] F. McNeill, A. Bundy, and C. Walton. Facilitating agent communication through detecting, diagnosing and refining ontological mismatch. In *Proceedings of the KR2004 Doctoral Consortium*. AAAI Technical Report, in press.
- [2] Maria Fox and David Long. An extension to PDDL for expressing temporal planning domains. Available from Durham Planning Group webpage: <http://www.dur.ac.uk/computer.science/research/stanstuff/planpage.html>.
- [3] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Stanford, CA, USA, 1992.
- [4] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18:1:1–31, 2003.
- [5] Drew V. McDermott, Dejing Dou, and Peishen Qi. An automatic translator between pddl and daml. [http://www.cs.yale.edu/homes/dvm/daml/pddl\\_daml\\_translator1.html](http://www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator1.html).
- [6] F. McNeill, A. Bundy, and C. Walton. Diagnosing and repairing ontological mismatches. In *Proceedings of the second starting AI Researchers' symposium*, Valencia, Spain, August 2004.
- [7] Fiona McNeill, Alan Bundy, and Chris Walton. An automatic translator from KIF to PDDL, December 2004. <http://planning.cis.strath.ac.uk/plansig/index.php?page=past22>.
- [8] 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>.
- [9] AKT Project. <http://www.aktors.org>, 2002.
- [10] PSL. <http://www.mel.nist.gov/psl/>.
- [11] SUMO. <http://ontology.teknowledge.com/>.