Dynamic Ontology Refinement

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

Abstract

One of the main reasons why plan execution meets with failure is because the environment in which the plan is being executed does not conform to an agent's expectations of how it will behave. If an agent is forming and attempting to execute plans in a given domain, these plans will be based on the agent's understanding of the domain, described in its ontology. Errors in this ontology are likely to lead to plans that are not executable. We propose to address this problem by dynamically refining the ontology as execution failure occurs and replanning using this updated ontology, thus creating more robust plans that are more likely to be executable.

Introduction

In order to generate plans that are sure to be executable in a given domain, it is necessary to have a complete and correct understanding of that domain. In all but the smallest of domains this is unfeasible, partly because the amount of resources necessary to fully explore and represent a large domain would be prohibitive, and partly because the domain may be dynamic and changing under external and possibly unpredictable influences. Thus the representation of the domain is often vague and imprecise and in some cases may be incorrect. In particular, the understanding of the domain is likely to be oversimplified, with the increasing detail of the domain becoming clear only through interaction with the domain. This is a normal occurrence in human understanding, where domains that are not well understood by a person are represented in vague, general terms. During interaction with that domain, the original representation seems not to be discarded but instead refined to include the additional information discovered during the interaction [7, 9].

We use the word *ontology* to refer to the whole of the agent's knowledge of a domain. This ontology could be shared or partially shared with other agents, or could be unique to a particular agent. Within the ontology, we distinguish between:

- the *signature*, which describes the types of things that exist in the domain; for example, listing all the predicates together with their arity and the type of the arguments they take;
- the *theory*, which describes the instantiations of the signature objects; for example, the specific things that exist and the rules that describe what actions can take place in the domain.

Lack of precision is likely to exist in both of these. In fact, vagueness in the signature entails vagueness in the theory, as all instances of this part of the signature in the theory will also be vague.

During plan execution, it is possible to learn further information about the domain. Failures in plan execution provide important information about what parts of the ontology are incomplete or incorrect, but also result in failure to attain the goal state. Thus we propose to dynamically refine the ontology using information about the domain that was learned during the attempted execution. We then use this refined ontology to create a plan that is more robust and more likely to achieve success. Much work on updating theories has been done in the related field of belief revision [2, 4, 5], and in some cases this will be similar to what we are doing. The difference lies in the fact that belief revision is concerned with semantics and considers beliefs to be unit objects, whereas we are interested in the syntax of a belief or ontological object. Also, belief revision is concerned with the facts themselves, whereas we are also interested in refining the signature of an ontology.

We believe that a casual understanding of a domain will lead to over-simplified ontological objects: for example, a single predicate where many are required; predicates with lower arity than necessary; rules with insufficient numbers of preconditions. When a richer understanding of the domain is developed through further interaction, these simple ontological objects should not be discarded and replaced, as occurs in belief revision, but instead should be refined to incorporate the extra detail.

We are interested in situations where the agent's ontology is close to a correct understanding of a domain but fails in parts. Our techniques are not applicable to a situation where two agents with completely different ontologies

are attempting to interact but rather to a situation where two agents have a similar understanding and comprehend much of what one another is communicating, thus making it possible to pinpoint a place in which disagreement occurs. We believe this is a reasonable situation to consider, since there are many cases where agents come from a common source and are later developed independently. One place this occurs often is on the Semantic Web, where the problem of ontological mismatch is significant.

The scenario we are interested in is that of a *plan-implementation* (PI) agent, attempting to execute a plan. The PI agent's role is to control the different components of the system and to execute the actions listed in the plan through interaction with other agents. This PI agent needs to have access not only to the plan and the ontology, but also to a *justification* of the plan, by which we mean a description of what rules and facts each plan step is based on, and an explanation for why they are believed to be true. This is necessary so that when a plan step proves to be inexecutable, the reason for this can be traced back to the ontology. This justification provides a set of ontological objects, either in the signature or theory, that may be at fault; for example, a rule and some preconditions for that rule with their justification. In order to find the exact point of failure, further agent communication may be necessary. The PI agent must question the agent it expected to perform the action of its plan as to why the action was not performed; the objects in the justification are queried and further questions asked until the point of failure is discovered. The point of failure is then passed to a refinement system, where the ontology is altered appropriately. At this stage, further questions may need to be asked in order to establish what the correct refinement is. For example, if the arity of a predicate needs to be increased, what would be the type of this extra argument?

In this paper we describe a dynamic ontology refinement system to correct failed plans, hence using the additional domain information learned during plan execution failure to create more robust plans. First we discuss how the point of failure is located through looking at the justification of the plan and also through further agent communication. Next we lay out how we patch this point of failure. We then outline the architecture for the system, provide a worked example and finally conclude the paper.

Locating the cause of failure

We believe that a sensible approach to the problem of planning with incomplete or incorrect information is to use an attempted plan execution to learn more about the domain, and hence develop a better domain description to facilitate the creation of better plans. This mirrors the way in which humans cope with the problem of vague or wrong knowledge. In the real world it is impossible to understand the complexities of every situation we may encounter, or the way these situations may have changed since we last encountered them. Humans seem to be very adapt at constantly refining their ideas and replanning in order to reach a goal that was unattainable using the initial plan. This is the kind of technique we are intending to emulate in this system.

The plan deconstruction

The first step in tracing the cause of plan-execution failure to a problem in the ontology is to examine which parts of the ontology were used to justify the plan step where failure occurred. For example, the justification for the action $buy(ticket)$ may be a rule such as:

Rule Name: $buy\text{-}rule$
Preconditions:

$$\begin{aligned} at(Place, Agent) \quad & \wedge \\ buy\_at(Thing, Place, Price) \quad & \wedge \\ has(money(X), Agent) \quad & \wedge \quad X > Price \end{aligned}$$

Effects: $has(Thing, Agent) \wedge$
$has(money(X - Price), Agent)$

together with an explanation for why the preconditions of this rule are held to be true. Note that the justification of a plan step explains why it should be possible to perform that plan step and not why the plan step is necessary. In order to establish where the fault is, we need to investigate why we thought it should have been possible to perform the plan step.

Justifications for a plan step will always be formulas from the theory of the ontology, since it is the specific beliefs about the domain that are required to form a plan rather than the general description of the types of things found in that domain. However, the underlying error may well occur in the signature of the ontology. For example, it may be that $money$ should be a binary predicate, with $currency$ being the type of the second argument. This problem would be highlighted by a failure to execute the buy action due to the precondition $has(money(X))$ being incorrectly stated. So the problem in the signature is revealed through a failure in the theory. Once the signature problem has been fixed, all occurrences of it in the theory need to be altered, including the occurrence that indicated the problem.

Since this kind of information is hard to extract from most planners, particularly state-of-the-art planners, we have developed a *plan deconstructor* to provide this information. This is loosely inspired by the plan validator developed at the University of Durham by Long and Cresswell [6], to validate plans by pseudo-executing them. Although the motivation is very different, the basic idea of justifying each plan step with reference to the ontology is the same. The plan deconstructor can be built on top of any planner. It takes the plan produced by this planner and deconstructs how it could have been built from the theory. It is not attempting to literally follow how the plan was built, it is instead using the theory to justify the plan that has been produced by the planner, in order that the justification

can be referred to when failure occurs. This means that the PI agent can narrow down the search for the problem to the relevant area. However, this information is not sufficient to identify a unique problem; there will still be many possibilities. If a rule is found to produce the wrong output it may be that the rule itself is incorrect; it may be lacking a precondition, for instance. Alternatively, one of its preconditions may be incorrect, and thus it is falsely believed that the rule is applicable. If there is an error in the theory, then the specific fact is incorrectly held to be true and we must search further back in the justification to see why we believed this to be true, thus locating the original source of the error. If it is due to a lack of precision in the signature, then we need to update the signature and alter all the occurrences of this predicate in the theory.

Plan deconstruction example Suppose the plan deconstructor is passed a plan such as the following one:

$[buy(ticket), walk(station, embassy, bishkek),$
$buy(visa), travel(bishkek, tashkent, bus)]$

together with the PI agent's ontology. The plan deconstructor annotates each action in the following ways: the rule used is given, together with a reason for why each of the preconditions are held to be true; for example, because it is a fact in the theory, or true in initial state, or true after action X. Also given are the effects of the action, i.e. what becomes true under that action, and the current status of the $fluents$, or predicates that change value depending on the situation (for example location and possession of items). A justification for the above plan as produced by the plan deconstructor is given below.

Action 1:	***buy(ticket)***	
Rule Used:	Rule 2	
Preconds:	$at(station, me)$	initial situation
	$buy\_at(ticket, station, 3)$	fact
	$has(money(10), me)$	fact
	$10 > 3$	true
Effects:	$has(ticket, me)$	
	$has(money(10-3), me)$	
Status:	$at(station, me)$	
	$has(money(7), me)$	
	$has(ticket, me)$	

Action 2:	***walk(station,embassy,bishkek)***	
Rule Used:	Rule 1	
Preconds:	$at(station, me)$	initial situation, status from Action 1
	$sit(station, bishkek)$	fact
	$sit(embassy, bishkek)$	fact
Effects:	$at(embassy, me)$	
	$\neg at(station, me)$	
Status:	$at(embassy, me)$	
	$has(money(7), me)$	
	$has(ticket, me)$	

Action 3:	***buy(visa)***	
Rule Used:	Rule 2	
Preconds:	$at(embassy, me)$	result from Action 2
	$buy\_at(visa, embassy, 2)$	fact
	$has(money(7), me)$	result from Action 1
	$7 > 2$	
Effects:	$has(visa, me)$	
	$has(money(7-2), me)$	
Status:	$at(embassy, me)$	
	$has(visa, me)$	
	$has(ticket, me)$	
	$has(money(5), me)$	

Action 4:	***travel(bishkek,tashkent,bus)***	
Rule Used:	Rule 3	
Preconds:	$at(embassy, me)$	results from Action 2, status from Action 3
	$sit(embassy, bishkek)$	fact
	$transport(bishkek, tashkent, bus)$	fact
	$has(ticket, me)$	result from Action 1, status from Actions 2,3
	$has(visa, me)$	result from Action 3
Effects:	$at(tashkent, me)$	
Status:	$at(tashkent, me)$	
	$has(visa, me)$	
	$has(money(5), me)$	
	$\neg has(ticket, me)$	

Agent communication

As described above, the plan deconstruction alone is not enough to find the point of failure. It is only possible through further investigation for the PI agent to decide exactly what part of the ontology to send to the refinement system. The PI agent cannot know exactly why the action failed; however, the agent with which it is communicating — for example, a *ticket-selling* (TS) agent — must have had a reason for failing to perform the action and must therefore have information about what the problem point is. The TS agent will have refused to perform the action because some part of his ontology prevents the action being possible; the corresponding part of the PI's ontology suggests that it should be possible. Hence this is where the ontology mismatch occurs, and if the PI agent wishes to interact successfully with the TS agent, he must alter this part of his ontology accordingly. Note that we are interested in agents that have similar but slightly differing ontologies; for example, agents whose ontologies have originated from the same source but have been developed separately. The PI agent is thus persuaded to refine its ontology to bring it in line with the agent with which it needs to interact, at least for the fragment of the ontology that is relevant to the action the PI agent wanted to execute. It may that there are some parts of the PI agent's ontology which it is not prepared to refine. In this case it will have to look for another agent that has different requirements to perform the same action.

An example of the kind of communication that may occur is given below. In this case, the PI agent has a

different ontological representation of the concept of money to the TS agent. The PI agent has money represented as a unary predicate taking a numerical value as an argument, the TS agent, instead, as a binary predicate taking a numerical value and a currency as arguments. This could occur because the PI agent has only ever dealt with one currency, and thus does not feel the need to specify, whereas the TS agent may be working in a domain where different currencies are used.

PI: $buy(ticket)$
TS: $has(money(P, dollars))$?
PI: fails $has(money(P, dollars))$
TS: fails $buy(ticket)$

In this case the cause of the problem is immediately clear; the PI agent has been confronted with a concept of money different to the one currently in his ontology and must refine his ontology accordingly.

If the cause of the problem is in the signature of the ontology then it is immediately apparent, since the PI agent is confronted with predicates it does not understand. If the problem is with the theory of the ontology, it is more difficult to discover and further communication is necessary. For example, the PI agent may have a fact $buy\_at(ticket, station, 10)$, whereas the TS agent may have a fact $buy\_at(ticket, ticket\_office, 10)$. Thus when it comes to check the preconditions for its selling rule it finds that the PI agent is in the wrong location and the conversation goes as follows:

PI: $buy(ticket)$
TS: fails $buy(ticket)$

The PI must then investigate further by checking which of the preconditions of the rule the TS agent does not agree with. If all the preconditions are accepted then the rule itself must be at fault.

Agent communication system Controlling agent communication is an important aspect of the system. We need to define what kind of speech acts are acceptable, whilst at the same time not restricting this too tightly, so that communication is possible between agents with somewhat different ontologies. In order to do this our system is based on the idea of eInstitutions [8, 10, 11]. We briefly describe these below, and then outline how we adapt these to our purposes.

An eInstitution consists of a set of predefined $scenes$, and $transitions$ between these scenes. A scene describes what kind of $illocutions$, or speech acts, can occur within it, and thus what actions can be performed here, and are controlled by institution or admin agents. External agents entering the institution must take on $roles$, for example as a $buying\text{-}agent$ or a $selling\text{-}agent$. So for the example situation described above, we would need to have a scene where the buying and selling of tickets was facilitated, plus scenes allowing whatever other actions we would wish to be performable. The institution agents ensure that the scenes, transitions and the institution itself run smoothly. They do not themselves participate in the buying or selling activity, or other function of the institution. The agents who would, for example, sell tickets, would be external agents who visit the institution because they wish to sell tickets. The PI agent will have to rely on the presence of the agents he wishes to interact with before he can achieve his goal within a scene.

This type of organisation creates a forum for our agent interaction. This is how we control how a PI agent would find the agents with which he needs to interact, and how, using the institution scene agents, this interaction is controlled. In the agent's ontology, in addition to the domain information, he has information about which scenes exist, which agents he can expect to find in which scenes and how he can move between these scenes.

Refining the problem point

Once the exact source of the problem has been located, it is sent to the refinement system to be corrected. At this stage, further communication with the other agent is likely to be necessary. Since we are mostly concerned with adding details, we are likely at this stage to require more information about this extra detail. For example, if we need to add an argument to a predicate, we will need to know what the type of the argument is.

Much work has been done in the field of removing detail from theories (abstraction). Hence, in order to formulate techniques for adding detail, we have investigated common abstraction techniques and considered how to invert these. Walsh and Giunchiglia claim that almost all abstractions fall into four categories [3]:

1. **Predicate abstractions**
   mapping predicate names in some uniform way:
   e.g. $bottle(x)$, $cup(x)$ map onto $container(x)$.
2. **Domain abstractions**
   mapping constants and function symbols in some uniform way:
   e.g. $prime(3)$, $prime(5)$ map onto $prime(oddnumber)$.
3. **Propositional abstractions**
   dropping some or all of the arguments to predicates:
   e.g. $abelian(groupA)$, $abelian(groupB)$ map onto $abelian$.
4. **Precondition abstractions**
   mapping some of the atomic formulas onto true or false:
   e.g. $has(ticket, me) \rightarrow can\text{-}travel(me)$ maps onto $can\text{-}travel(me)$

From these four types of abstraction we have developed four kinds of *anti-abstractions*.

Signature Refinement

Three of these anti-abstractions are signature refinements of the ontology:

1. **Predicate anti-abstractions**
   A single predicate is divided into some number of sub-predicates:
   e.g. $money(X)$ maps onto $dollars(X), euros(X), sterling(X)$
2. **Domain anti-abstractions**
   Constants and function symbols are divided up into different cases:
   e.g. $money(X, european)$ maps onto $money(X, euros), money(X, sterling),$ $money(X, krona)$
3. **Propositional anti-abstractions**
   Extra arguments are added to predicates:
   e.g. $money(X)$ maps onto $money(X, dollars), money(X, sterling)$

Once the refinement has been made to the signature, this always requires some change in the theory. Since the error in the signature has led to plan failure, we know that there must be at least one occurrence of the signature symbol in the theory that was the cause of this failure. That, and every other occurrence of the signature symbol, needs to be altered. This, in most cases, requires further agent communication in order for the system to be able to use the new version of the signature object. In some cases, where it occurs in parts of the theory that are not related to the agent that is currently being communicated with, it may not be possible to immediately instantiate the occurrence of the signature symbol in the theory, for example, if a new argument is added it may not be immediately clear what value this object may take, and it may be left uninstantiated until it is necessary to instantiate it.

Theory refinements

The theory has two components: rules and facts. One of the anti-abstractions describes a method for refining rules by adding preconditions:

- **Precondition anti-abstractions**
  Preconditions can be added to rules:
  e.g. the preconditions $has(invitation)$ could be added to the buy rule in order to purchase a visa:
  Rule-name: $buy\text{-}rule$
  Preconds:
  $$\begin{aligned} at(Pl) &\quad \wedge \\ buy\_at(Thing, Pl, P) &\quad \wedge \\ has(money(X)) &\quad \wedge \\ X > P &\quad \wedge \quad has(invitation) \end{aligned}$$
  Effects: $has(Thing) \wedge has(money(X - P)$

In most cases, facts will be refined due to signature refinements. However, in some cases we may find facts causing errors not because they lack detail but because they are missing entirely, or simply wrong. This may happen in a dynamic environment, where facts need to be updated. This kind of alteration is not a refinement but a kind of belief revision. Although this is not central to our interests since it is not refinement, it is may well cause problems and we will need tactics to deal with it.

Refinement example

In the following example, we discuss situations where these refinements are used. The agent communication aspect is not discussed in detail; for more information on this, see the Agent Communication section.

Suppose the PI agent's next step in the plan is to perform an action $buy\text{-}ticket$ and this is justified by the following rule in the agent's ontology:

Rule-name: $buy\text{-}ticket\text{-}rule$
Preconds:
$$\begin{aligned} at(X, Agent) &\quad \wedge \\ has(money(A), Agent) &\quad \wedge \\ transport(X, Y) &\quad \wedge \\ has(visa(Area), Agent) &\quad \wedge \\ cost(X, Y, C) &\quad \wedge \quad P > C \end{aligned}$$
Effects: $has(ticket(X, Y), Agent) \wedge$ $has(money(A - C), Agent),$

which, in this case, is instantiated as follows:

Preconds:
$$\begin{aligned} at(bishkek, me) &\quad \wedge \\ has(money(10), me) &\quad \wedge \\ transport(bishkek, tashkent) &\quad \wedge \\ has(visa(CIS)) &\quad \wedge \\ cost(bishkek, tashkent, 5) &\quad \wedge \quad 10 > 5 \end{aligned}$$
Effects: $has(ticket(tashkent, bishkek), me) \quad \wedge$ $has(money(5), me).$

He approaches the appropriate agent and asks him to perform the $buy\text{-}ticket$ action. The TS agent will have a rule in his own ontology which describes the conditions under which a ticket can be bought, in this case of the form:

Preconds:
$$\begin{aligned} at(bishkek, Agent) &\quad \wedge \\ has(money(A, dollars), Agent) &\quad \wedge \\ has(invitation, Agent) &\quad \wedge \\ has(visa(uzbek), Agent) &\quad \wedge \\ bus(bishkek, tashkent) &\quad \wedge \\ cost(bishkek, tashkent, 5) &\quad \wedge \quad A > 5 \end{aligned}$$
Effects: $at(tashkent, Agent) \wedge$ $has(money(A - 5), Agent)$

Some of these preconditions are fully instantiated already and he can check them himself, for example $bus(bishkek, tashkent)$. Others are not instantiated initially, and he may have to check these with the agent he is communicating with, for example $has(money(A, dollars), Agent)$. He can immediately instantiate the variable $Agent$ to the name of the agent he is communicating, but he cannot tell what A might be until he questions the agent further. Additionally, he needs to check some predicates which are fully instantiated: for example, $has(invitation, Agent)$ will not contain uninstantiated variables once he has inserted the name of the agent, but its veracity cannot be determined without questioning the

agent.

In order to decide whether he can comply with the request, he first checks the instantiated predicates. Since he can find out that $bus(bishkek, tashkent)$ is correct, he does not need to question it, so although there is an ontology mismatch here, this does not come to light. Next he asks how much money the agent has. Since the PI agent is confronted with a predicate that is not of the form he expects, he passes this to the refinement system. Here, $has(money(A), Agent)$ is refined to $has(money(A, Z), Agent)$, where Z is an extra argument of unknown type. Further information about this new argument comes from the PI's ontology, if he already knows type information about the object $dollars$. Otherwise, he needs to question the TS agent to discover this information.

Next, the TS agent questions the truth of the precondition $has(invitation, Agent)$. The PI agent responds negatively to this response, and so the TS agent refuses to perform the task. The PI agent infers from this that $has(invitation, Agent)$ is a necessary precondition (this is not certain but is a sensible guess), and the refinement system performs *precondition anti-abstraction* on the rule.

Finally, the TS agent questions the $has(visa(uzbek), Agent)$ precondition. The PI agent must again reply negatively to this, leading to failure, but in this case the question is not unexpected, merely differently instantiated. If the PI agent has the right information, he will be able to tell that $uzbek$ is a country within CIS, and thus the refinement system performs *domain anti-abstraction*. If he does not have this information, he extracts it from the TS agent.

At this point, the TS agent performs the action, even though there is still an ontological mismatch: $bus(bishkek, tashkent)$ matching $transport(bishkek, tashkent)$. This is acceptable because we are interested in updating ontological problems that affect plan execution. It is likely that this ontological mismatch will cause problems at another point, at which stage it is refined using *predicate anti-abstraction*. If it never causes problems then the ontological mismatch remains.

Worked Example

Worked examples for different aspects of the system are given in the relevant sections. Here, we briefly tie together these worked examples to describe how the whole system works.

Firstly, the PI agent forms the plan:
$[buy(ticket), walk(station, embassy, bishkek),$
$buy(visa), travel(bishkek, tashkent, bus)]$.

This is sent to the plan deconstructor, which annotates it as previously described and returns it to the PI agent.

The PI agents next attempts to execute the first action of the plan, $buy(ticket)$. He communicates with a $ticket\text{-}selling$ (TS) agent, and meets with plan failure. In this case there is an obvious clue about the cause of the failure, namely that the TS agent has a different signature for the object $money$. This is immediately obvious from the questioning of the TS agent, and in this case, recourse to the plan deconstruction is not immediately necessary. The $money(A)$ object in the signature is sent to the refinement system, where *propositional anti-abstraction* is used to refine it to $money(A, Currency)$. Suppose further questioning of the agent results in the object $has(money(10))$ being refined to $has(money(10, sterling))$ in the theory of this ontology. Once the refinement has been performed, replanning occurs, which in this case produce the following plan:

$[exchange(10, sterling, dollars), buy(ticket),$
$walk(station, embassy, bishkek), buy(visa),$
$travel(bishkek, tashkent, bus)]$.

Note that in some cases the plan looks the same as the original plan; for example, if the PI agent discovers that the currency of his money is already dollars. Then no extra action is required. However, the new plan is executable where the old one was not, because the ontology it is based on is more accurate.

Recourse to the justification from the plan deconstruction becomes necessary when no clues are given by the other agent; for example, in the second situation described in the Agent Communication section. Here, the TS agent simply refuses to perform the action. The PI agent must turn to the justification, as detailed in the plan deconstruction example, where he sees that this action is based on rule 2. He queries each of the preconditions of that rule. If one of these proves to be at fault, he must look at its justification: perhaps it is incorrectly stated in the theory of the ontology, or perhaps it was made true by some other rule, and in that case the PI agent must look further back for the cause of the failure.

Architecture

The system is controlled by a central *plan implementation* (PI) agent. The PI agent has access to the ontology that describes the domain. These are sent to the planner, in a suitable translation (for example, PDDL [1]), together with the goal the PI agent wants to achieve. The planner outputs a plan and this, together with the original ontological theory, is sent to the plan deconstructor, which will return the plan, annotated with a justification, to the PI agent. At this stage the PI agent attempts to execute the plan through communication with other agents. If failure occurs, the PI agent examines the justification developed by the plan deconstructor and then, through further communication with the agent he is interacting with, pinpoints the exact point of failure. This is then sent to the refinement system, where a decision is made about how to refine the problem and further communication with other agents, via the PI agent, is likely to

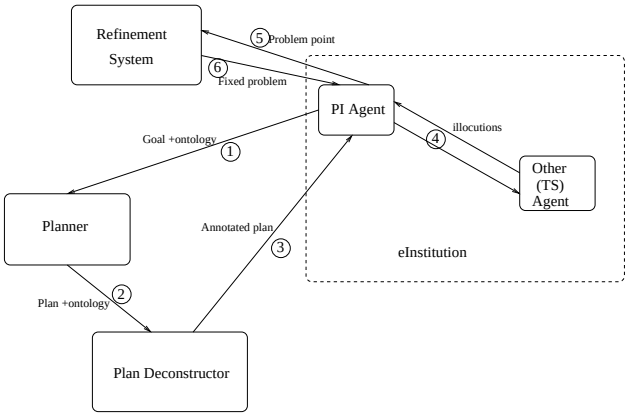

Figure 1: Architecture and interaction of a dynamic ontology-refinement system

occur. The refinement is then sent back to the PI agent, who deletes the original problem from the ontology and replaces it with the refined version. The process is then repeated with the updated ontology, and a more robust plan is developed.

Conclusions and further work

We have identified the problem of ontological mismatch as a major factor in plan execution failure. We have presented a way in which these ontological mismatches can be identified by *plan-implementation* agents and then refined. This refined ontology can then be used to produce a new plan which stands a better chance of being executable. As many plans are executed in the same domain, the understanding of that domain will become more sophisticated and accurate, and further plans created for that domain will be more likely to be executable.

So far we have developed the plan deconstructor and the refinement system, and we have decided what these refinements are and how they work. Our next focus is the agent communication system, and we are focusing on eInstitutions as our communication framework. Due to the kind of speech acts we require, we expect that additional work beyond eInstitutions may need to be done.

References

[1] 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`, 2002.

[2] P. Gardenfors and H. Rott. Belief revision. In D. Gabbay, C. J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 35–132. Oxford Science Publications, 1995.

[3] Fausto Giunchiglia and Toby Walsh. The use of abstraction in automatic inference. 1990. Proceedings of UK Conference on Information Technology (IT-90).

[4] P. Langley, G. Drastal, R. Bharat Rao, and R. Greiner. Theory revision in fault hierarchies. *The Fifth International Workshop on Principles of Diagnosis*, October 1994.

[5] D. Ourston and R.J. Mooney. Changing the rules: A comprehensive approach to theory refinement. *Proceedings of the National Conference on Artificial Intelligence*, 2:815–820, 1990.

[6] Plan validator. `http://www.dur.ac.uk/d.p.long/IPC/resources.html`.

[7] G. Polya. *How to Solve it*. Princeton University Press, 1945.

[8] Carles Sierra, Nick R. Jennings, Pablo Noriega, and Simon Parsons. A framework for argumentation-based negotiation. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Fourth International Workshop on Agent Theories Architectures and Languages (ATAL-97)*, volume 1365 of *Lecture Notes in Computer Science*, pages 177–192. Springer-Verlag, 1998.

[9] Alice ter Meulen. Representation and human reasoning. 2001. Third Augustus de Morgan Conference on Logic.

[10] W. Vasconcelos. Expressive global protocols via logic-based electronic institutions. Technical Report AUCS/TR0301, Department of Computing Science, University of Aberdeen, 2003.

[11] Wamberto Vasconcelos. Skeleton-based agent development for electronic institutions. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*. ACM Press, 2002. Part II.