Refinement in Agent Groups

Ciara Byrne and Peter Edwards
{byrne, pedwards}@csd.abdn.ac.uk

Department of Computing Science,
King's College,
University of Aberdeen,
Aberdeen,
Scotland AB9 2UE

Abstract. A group of intelligent agents may work together in order to solve a problem or achieve a common goal. If the group fails to achieve a goal, it may be able to adapt its behaviour so that such a goal can be achieved in the future. One of the ways in which the behaviour of the agent group can be changed is by refining the knowledge of individual agents. We are developing a distributed refinement system called DRAMA (Distributed Refinement Among Multiple Agents) to perform this task. The system makes use of a special type of agent called a refinement facilitator which coordinates the refinement process within the agent group.

1 Motivation

Suppose that a group of people want to cook Christmas dinner. They divide up the work in some manner: one person is to prepare the turkey, another to make the Christmas cake, etc. The goal of the group is to produce the meal at an appropriate time on Christmas day and this goal can only be achieved if all members of the group contribute. The person who is supposed to make the Christmas cake doesn't start preparing it until Christmas morning. He reads the recipe and discovers that he doesn't have all the required ingredients and since there are no shops open, he is unable to make the cake. In a different case, he may make the cake but find that he can't cook it because the oven is already being used to cook the turkey. Another possibility is that another member of the group sleeps in and therefore fails to carry out his task. These are some examples of how a group goal fails due to the actions of one or more of the participants. How can the same group of agents ensure that it will succeed if it attempts to cook Christmas dinner next year? In other words, how can the group ensure that it will be able to achieve a similar goal in the future?

A group of intelligent agents should be able to ensure that a failure is not repeated by adapting their behaviour. Adaptation can be achieved by using machine learning techniques to refine the knowledge of agents. The aim of learning in any intelligent system should be the improvement of the performance of that system. Where the system consists of a number of interacting intelligent agents,

performance may be evaluated by the coherence of the agent group or from the point of view of individual agents' success in achieving their goals. "Coherence will refer to how well the system behaves as a unit" [1]. Coherence may be measured along several dimensions including the quality of the solutions which the system produces, the efficiency with which solutions are produced and how gracefully performance degrades in the presence of failure or uncertainty.

This paper is concerned with failure-driven learning, i.e. learning is prompted by the failure of the agent group to achieve a goal. Performance is rated by solution quality in the sense that the agent group either finds a solution (achieves the group goal) or fails to find one (the group goal fails). The process of refining the knowledge of agents attempts to improve solution quality by allowing the agent group to avoid failures in the future.

2 The Refinement Problem

Techniques for refining the knowledge held in a single knowledge base have already been extensively investigated [2] [3] [4] [5]. The process of refining the multiple related knowledge bases of a group of cooperating agents presents unique challenges. In addition to their domain knowledge, social agents have knowledge that allows them to interact with others. Agents may represent knowledge in different ways. Since agents each have their own knowledge base, finding the failure point (or failure points) in the group's knowledge is a much more complex problem than in a single knowledge base. Refining one agent's knowledge may change its behaviour in a manner that affects other agents who will need to react to these changes in an appropriate manner. It may be also be desirable to maintain consistency between the knowledge bases of various agents.

2.1 The Knowledge of Agents

An agent which interacts with other agents needs two distinct types of knowledge: domain knowledge and social knowledge. The agent's domain knowledge concerns its problem-solving domain and environment. An agent's social knowledge allows it to interact with other agents. This knowledge may include the following:

Communication: Knowledge about the physical means of inter-agent communication (e.g. information about communication channels and the locations of agents) and an inter-agent communication language.

Interaction: How to interact with other agents in order to procure services, perform tasks for others, etc. An agent may, for example, know how to use a cooperation strategy such as the Contract Net Protocol [6]).

Agent Models: An agent can use these models to identify other agents with whom it is useful to interact, and to make this interaction more effective. For example, an agent may wish to determine which agents have the skills necessary to perform a particular task. A model may contain information such as the skills of other agents and their level of authority in the group.

2.2 Failures

Agents may have different knowledge bases and disparate views of their environment. Since social agents interact with other agents as well as the environment, the possibilities for failure increase. There are three basic phases in the process of achieving a group goal: task decomposition, task allocation and the coordination of tasks while they are being carried out. Failures may occur if agents commit errors during any of these phases. Task decomposition may be incorrect or insufficient. In the case of insufficient decomposition, the agent to whom the task is allocated may have the skills necessary to perform the task but cannot relate the task description to these skills. For example, if one agent is allocated the task "make a Christmas cake", that agent may be able to measure ingredients, mix them, use the oven, etc. but does not have a recipe for Christmas cake. Task decomposition and allocation may not take into account dependencies between tasks, e.g. the ingredients for the meal must be purchased before the cooking can be done. Failures may also be caused by allocating a task to an agent which cannot perform it or by failing to allocate a task redundantly when agents are unreliable. During the execution of subtasks, an agent may fail to complete a task. This may prohibit another agent from performing its task, e.g. by using a cooking utensil which that agent needs and not returning it. An agent may fail to share important information in a timely fashion, e.g. by not informing other agents that it has finished using the oven (if this would be appropriate). These are just some examples of the new range of failures that are possible when a group of agents attempt to achieve group goals as opposed to individual ones.

2.3 Faults in Knowledge Bases

Any of the failures described above are the result of faults in the knowledge of one or more agents in a group. Faults in a knowledge base include incorrectness, incompleteness, inconsistency, redundancy and intractability. The meaning of these terms may require some reinterpretation when applied to the collective knowledge of a group of agents rather than a single knowledge base.

Incorrectness: Some part of the knowledge is inaccurate. The question of context is important here as the same piece of knowledge may be correct in some contexts and not in others. As different agents may use the same knowledge in different contexts this issue is even more important in an agent system.

Incompleteness: Some necessary knowledge is missing. In the case of social agents, incompleteness in social knowledge is an additional problem.

Inconsistency: While this is a fault in the knowledge of a single agent, it may not always be necessary to maintain consistency between the knowledge bases of group members.

Redundancy: While redundancy in a knowledge base may not actually cause failures, it is usually desirable to make a knowledge base as compact as possible. On the other hand, redundancy is *necessary* in an agent group if the agents share a common environment or have similiar abilities.

Intractability: It is too expensive to achieve the goal using the current knowledge. For example, an agent group may use an interaction protocol which means that the group uses too many resources (e.g. time) when attempting to achieve a goal. In this case, social knowledge may need to be refined in order to allow agents to interact in a more efficient manner.

3 Designing a Distributed Refinement System

Refinement systems generally execute a cycle similar to the following (based on EITHER [2]): Recognise that a fault has occurred, locate the failure point (or postulate several possible failure points) and determine what refinements need to be made to the faulty knowledge. The existing knowledge and examples of the concept whose definition is considered to be faulty may be used to guide the choice of refinements. Examples of concepts defined in the knowledge base are classified as either positive (an example of the concept) or negative (not an example of the concept). Classification is usually performed by an oracle. An oracle may also be used to detemine when a failure has occurred and to help locate the failure point. We hypothesise that while oracles may exist for each individual agent, there may not always be a single oracle that can evaluate the knowledge of the entire agent group. As a starting point for designing any refinement system for multiple agents, it is therefore useful to consider a number of questions:

1. How will the system recognise that a failure has occurred?
2. How will the failure point be located?
3. Where do the examples originate and how are they classified?
4. What is the form of an example?
5. How are refinements generated?
6. If there are several possible refinements, how are the final refinement(s) chosen?
7. When the knowledge of one agent is refined, will other agents in the group need to have their knowledge updated in some way?
8. Will an autonomous agent accept refinements to its knowledge which originate from some external entity?

3.1 The Argument Against Centralisation

One possible solution to the refinement problem would involve collecting the knowledge of all agents and refining it in a similar manner to that employed for a single knowledge base. Of course, consideration would need to be given to the fact that the knowledge originated from several different agents. However, we would predict several problems with this approach. Agents may represent knowledge in different ways and still be able to cooperate. Therefore, the knowledge bases of individual agents would have to be translated into a common representation before refinement could occur and the refined knowledge retranslated into the agents' individual representations. How would knowledge be reassigned to agents once it had been refined in conjunction with the knowledge of other agents? How would the refinement system determine when inconsistency and redundancy between agent knowledge bases is necessary and when it is a problem? If the knowledge of different agents is contradictory how can it be determined which is correct? In addition, a centralised refinement system would require access to the internal structure of agents and this may not always be desirable. In conclusion, we feel that this would constitute a centralised solution to a distributed problem and we consider a distributed solution to be more appropriate.

3.2 A Distributed Approach

We believe that a group of agents can determine the causes of a failure and implement an effective set of refinements if they cooperate by sharing their knowledge and different perspectives on a failure. Our refinement system DRAMA (Distributed Refinement Among Multiple Agents) consists of two parts: a refinement module within agents and a refinement facilitator. Agents use the refinement module to generate refinements to their own knowledge. This can be regarded as an extra skill which agents possess in addition to the skills which they use in problem-solving. The facilitator coordinates the refinement process within the group but does not itself generate refinements [1]. Agents recognise that a failure has occurred and inform the facilitator which then gathers refinement proposals from relevant agents. The facilitator evaluates the proposed refinements and chooses a subset for implementation; in other words it acts as a filter for refinements.

We believe that this approach has several advantages. It allows agents to retain their autonomy by letting them propose refinements to their own knowledge rather than having refinements imposed upon them. Difficulties arise in the refinement process because individual agents only have a limited view of any failure which arises in the achievement of a group goal. The facilitator attempts to solve this problem by considering the views of all agents who may have contributed to the failure. The capabilities of the facilitator could possibly be distributed among the agents in the group but the use of a single facilitator agent reduces

[1] One of the problem-solving agents in the group may also act as a facilitator.

the amount of inter-agent communication and negotiation that would otherwise be necessary.

4 GOAL

We decided to test this approach to refinement by applying it to agents written in an agent-oriented programming language called GOAL (Goal-Oriented Agent Language). Agent-oriented programming [7] is a new programming paradigm which attempts to use mentalistic concepts such as beliefs, desires and intentions to formally describe the properties of agents. GOAL is based on Agent-K [8]. An Agent-K agent is specified in terms of its capabilities, a set of initial beliefs and a number of commitment rules. An agent's *capabilities* are the actions which it can perform. A *belief* is a logical statement which the agent "believes" to be true at a particular point in time. An agent's beliefs may change over time. When an agent decides to carry out an action, it forms a *commitment* to do so. A commitment to perform an action may be made in response to a request from another agent. One of the ways in which commitments can be formed is by the firing of a commitment rule. A commitment rule's conditions are matched against incoming messages and the agent's current mental state. If the rule fires, then a commitment is formed to perform the action requested by the message sender. Agents use KQML (Knowledge Query and Manipulation Language) [9] messages for inter-agent communication.

4.1 The Test Domain

We have implemented a simple hunter-prey scenario using agents programmed in GOAL. We chose this domain because it has previously been used in the DAI literature [10] and both agents and their environment can be defined at various levels of complexity. During experimentation, the environment may be made progressively more realistic and complex, thereby making it more difficult for agents to achieve their goals. The domain has the characteristics that it is dynamic, ongoing and unpredictable. The aim of refinement is the improvement of the performance of the agent group. In our test domain, performance is measured by the quality of solution produced. In the simple scenarios discussed here, there are only two possible evaluations of solution quality: either a solution is found, i.e. the group goal is achieved, or it is not. Agents are hunters who seek and kill prey. Some hunters have the ability to initiate cooperation in order to achieve goals such as the killing of a large prey by a group of hunters. Unless they are involved in cooperation, hunter agents execute a very simple cycle of actions: find a prey, move to the prey if it is not in the current location, attack the prey and eat it. A special world agent simulates the environment by periodically updating agents with new sets of beliefs about the environmental state. How much of the world a hunter can "see" depends on a vision-range parameter which is set in the world agent. We expect this test scenario to become more complex as our work proceeds.

4.2 The GOAL Language

In addition to beliefs, capabilities and commitment rules, GOAL also allows an agent's individual goals to be specified in the form of a simple goal tree. The structure of a GOAL agent is shown in Fig.1. High-level goals are decomposed into subgoals and eventually primitive actions. Some goals require the cooperation of other agents. These group goals are described by the number and types of agents needed to achieve them and the actions which each of the agents will be required to take. When an agent wants to achieve a group goal, it requests the cooperation of appropriate agents. If they agree to participate, it sends them instructions during cooperation and dissolves the group either when the group goal has been achieved or cannot be achieved. All agents except the agent which initiated cooperation suspend their own goals during cooperation. As a result, agents must either be altruistic or believe implicitly that they will benefit from cooperation.

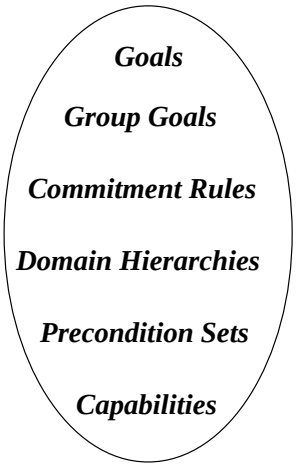

Fig. 1. A GOAL Agent

Associated with each action are various preconditions which must be satisfied before it can be performed. Two types of preconditions are defined: a *constraint set* and a set of *immediate preconditions*. The constraint set includes constraints on the characteristics of the agents with which the agent is currently cooperating (Wholist); when (When) and why (Why) the goal or action is being performed; and the arguments of the action (Arglist). Since there may be several precondition sets associated with a particular action, an index number is used to distinguish between them. A constraint set thus has the form given in Fig.2.

The *precond* field here consists of the set of immediate preconditions associated with the constraint set. The constraints are evaluated when an action or goal is under consideration. If a constraint set is satisfied, then the agent forms an instantiation [2] of the goal or action and forms a commitment to perform it.

```
constraints(Action,
            When,
            Arglist,
            Wholist,
            Why,
            Index,
            precond(Arg-names, Precond-sets))
```

Fig. 2. The syntax of a precondition set

Even when an agent has decided to take an action, additional preconditions concerning the mental state of the agent and the state of the world may need to be evaluated immediately before the execution of an action. We call this type of precondition an *immediate precondition.* The total precondition set shown in Fig.3 means that *Hunter1* will not form an instantiation of the action *eat(X)* unless its argument is an object of type *prey* and the action is being requested by another agent. Whether the agent is cooperating or not is irrelevant. The immediate preconditions indicate that the agent will not attempt to execute the action unless it is in the same position as the prey and the prey is dead. The reason for this separation of constraints and immediate preconditions is to differentiate the factors taken into consideration when making a commitment to perform an action, and the world state immediately before the action is attempted. The world state may change in the time between the formation of the commitment and its execution.

An agent may also have knowledge in the form of domain hierarchies. A hierarchy will describe a particular aspect of the agent's task or environment. For example, in the hunter-prey domain there may be a hierarchy which defines the prey in the environment as either small or large prey. Hierarchies are used to evaluate constraints, e.g. the constraint set of an action such as *attack(Object)* may contain a constraint which specifies that *Object* should be a small prey.

5 Refinement Within Agents

Agents propose refinements to their own knowledge. In theory, refinements could be made to any of the types of knowledge which GOAL agents possess, i.e. definitions of goals and group goals, domain hierarchies, capabilities and commitment

[2] Examples of instantiations are given later in this paper.

```
constraints(eat,
            time (−),
            [prey(−)],
            −,
            other-request(−),
            1,
            precond([Prey],
                    [[believes(hunter1, [Now, object-position(hunter1, Here)], t),
                      believes(hunter1, [Now, object-position(Prey, Here)], t),
                      believes(hunter1, [Now, object-status(Prey, dead)], t)]])).
```

Fig. 3. Example precondition set from the hunter-prey domain

rules. The form which a refinement will take depends on the type of knowledge being refined. For example, knowledge about capabilities may be changed by adding or deleting a belief about the capabilities of a particular agent, whereas the definition of a group goal may be modified by reallocating an action to an agent or increasing the number of cooperating agents. Initially, we have given agents the ability to refine precondition sets. The agent gathers information relevant to the fault in order to guide refinement, determines which type of knowledge needs to be refined (if refinements to several types of knowledge are possible) and applies refinement operators in order to generate a refinement. In the case of preconditions, GOAL allows agents to record information about the circumstances in which actions have been taken in the past. In order to participate in the refinement process an agent needs to know the following: how to describe faults to the facilitator and understand descriptions of faults, how to generate refinements and describe them, and how to update its knowledge if modifications are required due to the refinement of another agent's knowledge.

To generate refinements to preconditions, agents need information about the context in which a particular action has been performed in the past. Records of the circumstances in which actions are taken, i.e. the evaluation of constraints and preconditions, are continuously generated during an agent's execution. Such a record is termed an *instantiation* and has the form given in Fig. 4. The instantiation of a successful action can be seen as a positive example and an unsuccessful one as a negative example. If there are multiple precondition sets for a particular action, we need to identify which set has been satisfied. *C-set* indicates the constraint set that was satisfied by this instantiation and *P-set* the set of immediate preconditions that was satisfied.

In general, existing refinement systems work by applying generalisation operators when the fault involves the misclassification of a positive example and specialisation operators when a negative example is misclassified. In this context, a positive example of an action is one which was successful, e.g. an attempt to take the action *move-to([1,2])* results in the agent's new position being [1,2]. A

inst(Action, When, Arglist, Wholist, Why, C-set, P-set, Evals)

inst(move-to, [2,8,95,9,20,3], [[1,3]], [], hunter1, 1, 2, [t,t,t])

Fig. 4. The syntax of an instantiation

negative example would be the execution of an action in inappropriate circumstances, e.g. *attack(X)* where X is another hunter. Because there are two distinct types of precondition (constraints and immediate preconditions) in our system, we use the following set of refinement operators:

1. Specialise an existing constraint set.
2. Delete a constraint set.
3. Generalise an existing constraint set.
4. Add a new constraint set.
5. Delete an immediate precondition from an existing set.
6. Add a new set of immediate preconditions.
7. Add an immediate precondition to an existing set.
8. Delete a set of immediate preconditions.

An agent may apply some combination of these operators in order to generate a suitable refinement. Although these refinement operators may be used in any domain, a number of characteristics of the hunter-prey domain will determine exactly how they will function and what strategy is used to control their application. For example, we are assuming that a hunter's example set will consist largely of positive examples with few, if any, negative examples of actions. Therefore, refinement operators must be able to use positive example sets effectively. Obviously, this condition may not apply in other problem domains. Therefore, evaluating the generality of results obtained in this particular domain may be problematic. However, from the perspective of the facilitator the manner in which operators generate refinements is an internal implementation detail of the agents. The important thing from the point of view of the agent group is that refinements *are* generated and that they are presented in an appropriate manner to the refinement facilitator.

6 The Refinement Facilitator

A facilitator coordinates interaction between agents. For example, KQML [9] communication facilitators are used to manage message traffic among other agents by routing messages to appropriate agents, providing buffering and translation facilities, etc. The task of the refinement facilitator is to coordinate refinement by processing refinement requests from agents, soliciting refinement proposals, choosing the refinements which will eventually be implemented and

informing agents of changes to the knowledge of other agents. Much of the facilitator's knowledge will be domain-independent because it is concerned with general issues such as types of refinements and the relationships between them. It will also require some domain-specific knowledge in order to work with a particular group of agents. The refinement facilitator requires the following types of knowledge:

Information about agents: Names and URLs (Universal Resource Locators) of agents.

Descriptions of faults and refinements: The facilitator needs to know how faults are described so that it can correctly interpret and process descriptions received from agents. Similarly, the facilitator must correctly interpret descriptions of proposed refinements.

Refinement types: Refinements can be divided into categories depending on the type of knowledge being modified, e.g. precondition sets or a definition of a group goal.

Fault decomposition: The facilitator must know how to use a fault description provided by an agent to identify possible failure points and solicit refinements from appropriate agents.

Relationships between refinement types: We have proposed a number of possible relationships between refinements: refinements can be equivalent, complementary or conflicting. Two different refinements may have the same effect as regards correcting a fault, but involve different changes to the knowledge of agents. These refinements can be considered to be equivalent. Refinements conflict if they negate each other's effects and complement each other if both (or several) are needed to repair a fault.

Rating refinements: If there are alternative sets of refinements which may be applied to correct a fault, then the facilitator needs to have some way of choosing between them. A rating scheme is used to calculate ratings for refinements and the refinement set that achieves the best overall rating is approved.

Knowledge update rules: When an agent makes changes to its knowledge, other agents may need to be informed. This may be necessary in order to maintain consistency between the knowledge bases of agents. In other cases, agents may need to be informed about a refinement because of the changes this will cause to the future behaviour of an agent. A knowledge update rule thus defines a list of agents which should receive messages describing the refinement which occurred.

7 An Example

To illustrate our ideas, we present an account of how DRAMA would rectify a simple fault from the hunter-prey domain. The domain knowledge in this scenario consists of hierarchies which classify agents and prey objects, locations, and reasons for performing actions. For example, *Hunter1* has the domain knowledge shown in Fig.5. The agent's environment is divided into squares, each of which are referenced by x and y coordinates. Each square is characterised by the dominant type of terrain in that square, e.g. marshland. Prey animals in the environment are classified as large or small. Finally, agents may perform actions under their own initiative or because of a request from another agent.

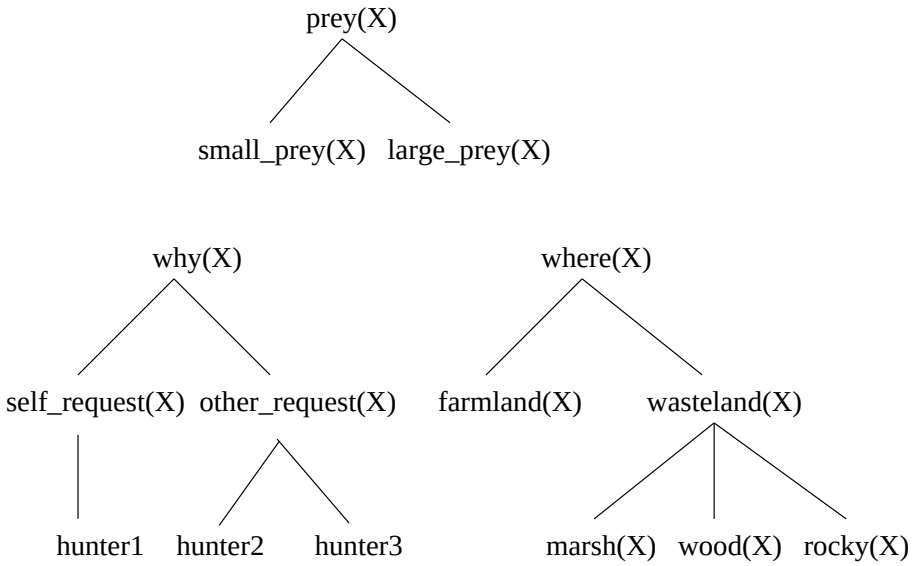

Fig. 5. Domain Hierarchies of Hunter1

All hunters have the ability to generate refinements that modify the precondition sets of particular actions. In addition, *Hunter2* can propose refinements involving changes to its beliefs about the capabilities of an agent. The simple refinement facilitator used in this example has the following knowledge:

Information about agents: *Hunter1, Hunter2, Hunter3* and their respective URLs.

Refinement types:
modify-preconds: An agent modifies the set of preconditions (either a constraint set or immediate preconditions) associated with one of its actions.
modify-capabilities: An agent modifies its view of the capabilities of another agent, e.g. by adding or removing an action from the list of actions that it believes another agent is capable of performing.

Relationships between refinements: If a *modify-preconds* refinement proposed by agent A is concerned with action X and a *modify-capabilities* refinement is concerned with the ability of A to perform X, the two refinements are considered equivalent.

Rating strategies: A refinement to a precondition set receives a better rating than a refinement to knowledge about capabilities.

Knowledge update rules: If the agent which initiated cooperation (i.e. the owner of the group goal) proposed a refinement of type *modify-capabilities* and it is accepted, inform other agents who were in the cooperating group of the change.

Hunter2 senses a large prey and requests the help of *Hunter1* and *Hunter3* in killing the prey. It sends the agents messages in which it proposes actions for them to perform. *Hunter2* does not specify details of actions at this stage, i.e. the time at which the actions are to be performed and their specific arguments. *Hunter1* and *Hunter3* reply indicating that they will participate. *Hunter2* then sends messages to *Hunter1* and *Hunter3* to confirm that cooperation will go ahead. *Hunter1* and *Hunter3* go into cooperation mode by suspending their own goals and await further instructions from *Hunter2*. *Hunter2* requests that they both move to the location of the prey. *Hunter3* moves to the location without difficulty. *Hunter1* tries to move through marshland in order to get to the prey with the result that it gets stuck and does not reach the prey. *Hunter2* recognises that the goal of killing the prey has failed because *Hunter1* did not move to the prey's location and dissolves the group. *Hunter2* sends a message to the refinement facilitator which names the agents involved in cooperation and describes the nature of the failure. In this case there is only one failure point. In more complex scenarios, there may be multiple failure points. The type of fault involved is the non-performance of an agreed action by *Hunter1*. The facilitator requests refinement proposals from both *Hunter1* and *Hunter2*. The facilitator will have determined from the failure description that *Hunter3* was not responsible for the failure since it carried out the requested actions.

As *Hunter1* can only generate refinements of type *modify-preconds*, it attempts to find a refinement of this type. On receipt of the request for refinement proposals, *Hunter1* examines its actions. Agents move from one location to another by executing a series of *next-location* primitive actions. *Next-location(Loc)* causes the agent to move one square in a given direction. *Hunter1* plotted a route

through marshland because this was the most direct route to its target location and the precondition set for next-location allowed it to do so. The total precondition set for *next-location(Loc)* is shown in Fig.6.

```
constraint(next-location,
           time(−),
           [wasteland(−)],
           −,
           why(−),
           1,
           precond([Loc], []))
```

Fig. 6. Hunter1's precondition set for the *next-location* action

Because this is an example of an action being performed in the wrong circumstances, the agent applies a specialisation operator. In this case, *Hunter1* does not know whether the constraint set or the set of immediate preconditions is at fault. The constraint set is specialised due to the existence of a heuristic which

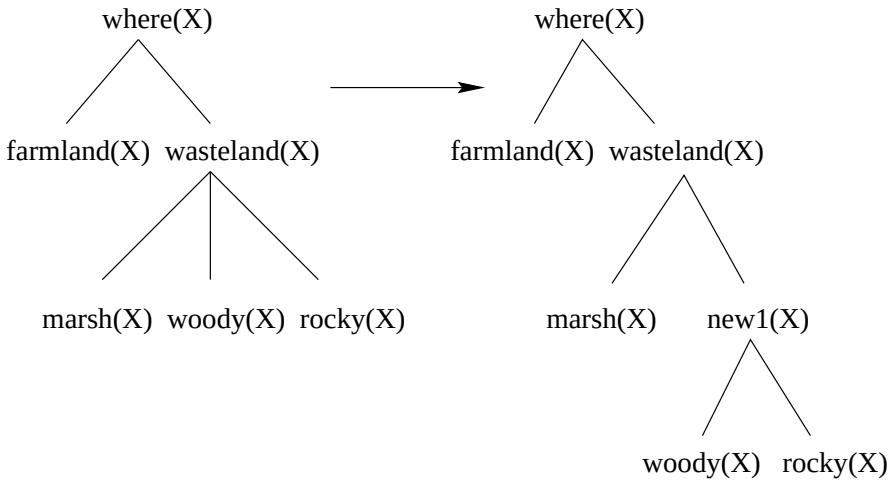

Fig. 7. Changes to the domain hierarchies

specifies that it is better to refine a constraint set than a set of immediate preconditions. This is because the agent has more information (i.e. evaluations of

constraint sets in previous actions) on which to base refinements of the former type. The refinement operator finds the first constraint in the set which can be specialised sufficiently to exclude the failing example. In this case, the *where* constraint of the *next-location* action is specialised sufficiently to exclude moves through marshland. The specialised constraint should allow the agent to move through any type of wasteland except marshland. The domain hierarchy for the *where* constraint must be changed in order to allow this. A new predicate, *new1*, is created that covers woodland and rocky areas but not marshland. The new domain hierarchies are shown in Fig.7. The new predicate replaces the old *where* constraint in the specialised constraint set (shown in Fig.8).

Having found this possible refinement, *Hunter1* suggests it to the facilitator. Since *Hunter2* has the ability to generate refinements of type *modify-capability*, it may suggest a change in its belief that *Hunter1* can move to a given location. The facilitator decides that the two refinements are equivalent. By using the rating strategy it chooses the refinement proposed by *Hunter1*. The facilitator informs *Hunter1* that it can carry out the refinement.

```
constraints(next-location,
            time(−),
            [new1(−)],
            -,
            why(−),
            1,
            precond([Loc], []))
```

Fig. 8. Hunter1's refined precondition set for the *next-location* action

8 Discussion and Future Work

8.1 DRAMA

To summarise the properties of our refinement system DRAMA (Distributed Refinement Among Multiple Agents), we will consider the questions raised in Section 3. To date, DRAMA only addresses faults related to incompleteness and incorrectness in precondition sets.

How will the system recognise that a failure has occurred? In general, the leader of a group goal will recognise that a failure has occurred because the goal has not been achieved. It will then inform the refinement facilitator that a failure has occurred.

How will the failure point be located? Agents analyse their own knowledge, postulate possible failure points and propose refinements to correct their knowledge.

What is the form of an example? Examples take the form of instantiations of actions.

Where do examples come from and how are they classified? Agents continuously gather examples of the use of actions. The agent observes the results of its actions and thereby classifies examples as successful or unsuccessful. Classification may also originate from an external source, e.g. the leader of a group goal (the agent which initiated cooperation).

If there are several possible refinements, how are refinements chosen? The facilitator uses its knowledge of the relationships between refinements in order to sort refinements into sets and then uses a rating strategy to choose the "best" set. In essence, it acts as a filter for refinements.

When the knowledge of one agent is refined, will other agents in the group need their knowledge updated in some way? Knowledge update rules fulfill this function by informing agents of changes to the knowledge of other agents. Agents which receive updates can use this information to change their own knowledge so that it remains consistent with the knowledge of others.

Will an autonomous agent accept refinements to its knowledge which originate from some external entity? Agents suggest refinements to their own knowledge and therefore refinements do not originate from an external source.

8.2 Evaluating Generality

Our ultimate aim is to design an approach to refinement that is generally applicable in two different senses; namely that it can be used in multiple domains and with agents which are written in diverse languages.

Agents work together in order to solve problems from a particular problem domain. A refinement scheme should be independent of any particular domain. We are currently designing a domain-independent framework for a refinement facilitator. This framework will need to be instantiated with some domain-specific knowledge in order to function with agents in a particular problem domain, e.g. the hunter-prey domain. The basic refinement cycle outlined above, the types of knowledge which a facilitator needs and the way in which the facilitator uses its knowledge should remain broadly the same regardless of the domain.

Designing a refinement system that is effective when applied to agents written in languages other than GOAL is a longer term objective, and may prove too complex to achieve. A language-independent facilitator is possible if agents can describe faults and refinements in a manner that is independent of the language in which they are written. Such a facilitator would need to be able to interpret these high-level descriptions.

8.3 Future Work

There are many issues, in addition to those outlined in earlier sections, which we plan to explore. These include the use of knowledge update rules, side-effect refinements, maintaining consistency between the knowledge of agents, more direct exchange of knowledge between agents, and increasing agents' input into the selection of refinements by the facilitator. A side-effect refinement is one which does not directly contribute to the resolution of the current failure but could allow the agent to avoid different types of failure in the future. Suppose an agent is asked to move to a particular location by the leader of a group goal. For some reason moving to this location has a detrimental effect on the agent. During the refinement phase it is determined that this action was not necessary in order to achieve the goal so it is removed from the definition of the group goal. However, it would also be beneficial to the agent to refine its precondition set so that it will not move to this location again. This is a side-effect refinement. Another type of side-effect refinement could occur when one agent modifies a precondition set for an action and other agents update their precondition sets with the refined one. Performance may also be improved by allowing agents to suggest refinements to each other's knowledge, provide confidence factors for refinements proposed by other agents and exchange information (such as instantiations) which can be used to guide the generation of refinements.

It should be emphasised that this is very much work in progress and our ideas will develop as we put them into practice. To date, we have concluded that agents need information on the context in which they take actions in order to generate refinements to precondition sets. We have made a first attempt to facilitate the continuous generation of such information by extending an existing agent language. We have implemented a basic hunter-prey scenario by programming a group of agents in this language. Our aims in the immediate future include building a prototype refinement facilitator and expanding the range of refinements which an agent can make. We also intend to test the refinement scheme in progressively more complex versions of the hunter-prey domain and eventually in a completely new domain.

References

1. A. H. Bond and L. Gasser, *An Analysis of Problems and Research in DAI*, Readings in Distributed Artificial Intelligence (1988), Morgan-Kaufmann, 3–35.
2. D. Ourston and R.J. Mooney, *Changing the Rules: A Comprehensive Approach to Theory Refinement*, Proceedings of the Eighth International Conference on Machine Learning (1991), 485–489.
3. G. Towell, J. Shavlik and M. Noordewier, *Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks*, Proceedings of the Eighth National Conference on Artificial Intelligence (1990), 861–866.
4. B.L. Richards and R.J. Mooney, *Learning Relations by Pathfinding*, Proceedings of the Tenth National Conference on Artificial Intelligence (1992), 723–738.

5. S. Craw and D. Sleeman, *The Flexibility of Speculative Refinement*, Machine Learning: Proceedings of the Eighth International Workshop (1991), 28–32.
6. R.G. Smith, *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*, IEEE Transactions on Computers, C-29:12 (1980), 1104–1113.
7. Y. Shoham, *Agent-Oriented Programming*, Technical Report STAN-CS-1335-90 (1990), Department of Computer Science, Stanford University.
8. W. Davies and P. Edwards, *Agent-K: An Integration of AOP and KQML*, CIKM Workshop on Intelligent Information Agents (1994), Y. Labrou and T. Finin (Eds), National Institute of Standards and Technology, Gaithersburg, Maryland.
9. T. Finin, R. Fritzson, D. McKay et al, *An Overview of KQML: A Knowledge Query and Manipulation Language*, Technical Report (1992), Department of Computer Science, University of Maryland.
10. M. Tan, *Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents*, Machine Learning: Proceedings of the Tenth International Conference (1993), 330–337.

This article was processed using the LaTeX macro package with LLNCS style