Motivating Intelligent Agents for Virtual Environments

Sorabain Wolfheart de Lioncourt and Michael Luck

Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK
Email: {bane, Michael.Luck}@dcs.warwick.ac.uk

Abstract. An agent with multiple requirements and limited or constrained resources must be able to make decisions as to how to divide those resources in order to best satisfy its requirements. It may not be possible to satisfy all of its requirements at once, so some requirements may have to be sacrificed for the sake of more important ones; in other cases a compromise may be possible in which all of its requirements are partially satisfied. This paper examines the kinds of requirements we may have of an agent, and how to measure the performance of an agent with respect to these requirements. Such a requirements analysis can be used as a conceptual design tool, and the basis of a specification of an agent. An agent architecture based on the belief-desire-intention model is proposed in which the design and implementation of an agent is decomposed in terms of its requirements. This architecture allows the intuitive development of agents with multiple requirements in dynamic virtual environments.

1 Introduction

While much work in the field of intelligent agents has sought to develop sophisticated agent architectures capable of reasoning and acting in addressing a range of tasks, it has largely ignored the issues involved in situating such agents in virtual environments. Similarly, work in virtual environments has tended to focus on lower-level agent capabilities more closely related to situatedness and embodiment. As these fields converge, however, the dividing line between them is beginning to be erased. In this paper we describe our work towards the development of a sophisticated agent architecture that extends an existing model so that it offers greater flexibility of control and is suitable for use in a dynamic virtual environment.

The base architecture is inspired by, and derived from an example of possibly the best-known class of agent architecture, the belief-desire-intention (BDI) model. dMARS [3] is an implemented and deployed commercial system that underlies this work, and is extended through the inclusion of mechanisms for motivated behaviour similar to artificial life approaches. The architecture is applied in a virtual city where emergency services must be coordinated to deal with situations of varying urgency, where agents play the rôle of a single emergency vehicle.

2 Directing Autonomous Agent Behaviour

An intelligent autonomous agent is expected to act for extended periods without human intervention. Although the agent is free to set its own goals and decide how best to

achieve them, we will have particular rôles for the agent in mind and will expect it to act accordingly. In this section we consider the kinds of behavioural requirement that we might place on an autonomous agent. We find that all can be considered as a special case of an *avoidance requirement*, and the transformation of any kind of requirement into its equivalent avoidance requirement is simple and remains intuitive. Transformation of all an agents requirements into a single type will enable us to consider the possible interactions between requirements more clearly, and allows us to implement an agent architecture that uses the same control system for all the kinds of requirements. To introduce the different kinds of requirements we consider a hypothetical virtual creature that must survive in an environment with scarce food and water, others creatures competing for food and water, and a youngster to protect.

2.1 Requirement Types

We first list the kinds of requirements we may wish to place on our agent. Initially the requirements simply specify the states of affairs we wish our agent to avoid or maintain. Later we discuss the relative *badness* of violating a requirement.

Avoidance Requirements The first type of requirement we consider is where the agent is required to avoid or maintain particular states of affairs over its whole lifetime. Such requirements could include maintaining homeostasis, or avoiding situations that put the agent in danger.

We first list these requirements in English. For example, a virtual creature might have the following requirements (among others) (see [9] for a full list of requirements a virtual creature might have):

- avoid death by dehydration;
- avoid death by starvation;
- avoid intruders encroaching on your territory;
- maintain proximity to cover; and
- maintain a good condition coat.

As we can see, the first two items are descriptions of states to be avoided and the last two are descriptions of states to be maintained. For consistency we transform all maintenance descriptions into the equivalent avoidance description. We simply replace a maintenance requirement, *maintain X*, with the avoidance requirement, *avoid complement(X)*. For the virtual creature we replace *maintain a good condition coat* with *avoid having a poor condition coat*, and *maintain proximity to cover* with *avoid being far from cover*. The final list is referred to as the *avoidance requirements* of the agent.

Periodic Requirements The second type of behavioural requirement is that of performing periodic behaviours. A periodic requirement is where we wish the agent to perform a particular task at regular intervals over its whole lifetime. Our virtual creature may be territorial and leaves scent markings around the borders of its territory, which need to be refreshed periodically. In addition to this, the creature might need to be vigilant to intruders encroaching on its territory.

We can treat a periodic requirement as a special case of an avoidance requirement. If we wish an agent to perform behaviour *X* every *T* seconds then we can form the avoidance requirement *avoid not having performed X in the last T seconds* and add it to the list of avoidance requirements. For our virtual creature we add the avoidance requirement *avoid not having performed scent marking in last 2 days*. To ensure that our agent is vigilant to intruders in its territory we add a requirement to scan its surroundings once every minute, modelling this with the avoidance requirement *avoid not having performed environment scanning in last 60 seconds*.

Social Requirements Avoidance and periodic requirements persist for the lifetime of the agent and are sufficient for an autonomous agent acting independently. However, the agent may be acting within a social group and is expected to consider performing tasks that benefit the group as a whole. An agent may choose to perform such tasks proactively, or it may be delegated tasks by other agents. We refer to any such task as a *social requirement*. Social requirements may be taken on for a limited period of time, or indefinitely.

A social requirement may be an avoidance or periodic requirement, or it may be a *one-off task* (see Section 2.1). The key difference between social requirements and normal requirements is that the social requirements of an agent may change over time as new ones are undertaken, or old ones are dropped, whereas the normal requirements of an agent persist for its lifetime.

An example of a social requirement in the case of our virtual creature may be to protect its young. Such a requirement may be considered a single social avoidance requirement, *avoid youngster being killed*, or perhaps several: *avoid youngster dying by dehydration*; *avoid youngster dying by starvation*; and *avoid youngster being killed by predation*. Such social requirements may be implicitly taken on by the parent at the time of birth, or may be delegated to some other agent. This requirement will persist until the youngster is deemed to be old enough to cope on its own.

One-off Tasks A special kind of social requirement is one in which an agent is delegated a one-off task, which is to achieve a particular state of affairs. Once achieved, the requirement is considered permanently satisfied, so that the agent is not required to maintain that state of affairs. The task may either be required to be completed as soon as possible, or before a given deadline. Both types of one-off task can be encoded as a special case of an avoidance requirement.

In the case where a task, *X*, is to be completed before a given deadline, *D*, we add the avoidance requirement *avoid D passed and X not completed* to the agent's current list of avoidance requirements. The case where a task is to be completed as soon as possible is a special case of a deadlined one-off task, for which the deadline is considered to be immediate. In both cases, once the task is completed the requirement is removed from the agent's set of requirements.

2.2 Requirement Violation

Ideally an agent will avoid the violation of any of its requirements, but in general this may not be possible where the agent has limited resources and different requirements

compete over them. Where an agent cannot avoid the violation of some of its requirements the designer may have some preferences as to which violations should be avoided with greater effort.

The preferences we have between violations of different requirements will partly depend on the nature of the violation, since not all violations have the same properties. Some violations can be rectified at a later date, while other violations may be permanent. In fact there are four kinds of violation that can occur.

Flexible Violation A flexible violation is one that can be rectified by an appropriate choice of actions. For example, violation of the *avoid being far from cover* avoidance requirement in our virtual creature is flexible because the agent can later choose to move towards cover once again.

Positive Trap Violation A positive trap violation is one in which rectification of the violation is permanent, and the requirement will remain satisfied for all time. A one-off task is an example of a positive trap violation.

Negative Trap Violation A negative trap violation is one which can never be rectified, and once violated it remains violated for all time. For example, violation of the *avoid youngster killed by predation* social avoidance requirement can never be rectified once the youngster has been killed.

Lethal Violation A lethal violation is similar to a trap violation in that it will remain violated for all time. But more than this, a lethal violation means that the agent becomes incapable of any action from the time of violation onwards. In the case of our virtual creature, violation of the *avoid death by dehydration* or *avoid death by starvation* avoidance requirements are lethal.

Typically a designer might expect the agent to do its upmost to avoid lethal or negative trap violations, preferring to tolerate a flexible violation or continuation of a positive trap violation. We describe a method of specifying such preferences in the following sections.

Lexicographic Preferences Given a list of requirements, the strongest kind of preference ordering we can impose is that of lexicographic preferences [8]. A pure lexicographic preference ranking of n requirements, $r_1, r_2, \ldots, r_n$ treats a requirement r_i as infinitely more important than any requirement r_j with $j > i$. In terms of violations, an agent will not tolerate any kind of violation (even a flexible violation of minimal duration) of a more important requirement in preference to any combination of violations of requirements lower in the lexicographic ranking. For example, we would not want our virtual creature to suffer death by starvation, no matter how much scent marking it can perform, or how well groomed it can keep itself. Note that a violation of an important requirement does not imply that all requirements of lower importance are also violated, or that more important violations are necessarily rectified before less important violations. It may be the case that a violation with lower importance can be rectified by using resources that the higher importance violation does not require.

Lexicographic preferences can be used to model default behaviours where we want our agent only to perform certain behaviours when it has nothing better to do. Grooming behaviour in animals is often considered a default behaviour [1] and so we consider it as

a default behaviour in our virtual creature. A default behaviour is placed after all non-default behaviours in the lexicographic preference ordering. If more than one default behaviour is present then we consider lexicographic dominance between them as for non-default behaviours.

In general we will not be able to produce a pure lexicographic preference ordering between the agent's requirements. Some requirements may have equal importance to others, such as avoiding death by starvation, and avoiding death by dehydration. In other cases we may wish to allow a trade-off between violations, where we consider one requirement to be more important than others, but we may tolerate a flexible violation of it if we can satisfy several less important requirements at once. In cases where no pure lexicographic preference exists between requirements we place them both on the same level in the lexicographic ordering.

The general lexicographic ordering we produce is between sets of requirements. A requirement is considered infinitely more important than those requirements in lower levels, and the current situation the agent finds itself in dictates the relative importance of requirements at the same level. An example of how requirements at the same level can trade-off is given in Section 5

For the virtual creature we came up with the following requirements.

- avoid death by dehydration (violations are lethal);
- avoid death by starvation (violations are lethal);
- avoid youngster being killed (social, violation is a negative trap);
- avoid having a poor condition coat (default behaviour, violations are flexible);
- avoid being far from cover (violations are flexible);
- avoid intruders encroaching on your territory (violations are flexible);
- avoid not having performed scent marking in last 2-days (violations are flexible); and
- avoid not having performed environment scanning in last 60-seconds (violations are flexible).

From these requirements we might produce the general lexicographic ordering shown in Figure 1. Since we have specified that grooming is a default behaviour it should clearly inhabit the least important level in the lexicographic ordering. Avoiding death by starvation and death by dehydration are both equally important, and we would not want to risk a chance of death by either in preference to scent marking, scanning the surroundings, chasing off intruders, or grooming. However, a strong parental instinct may tolerate an increase in the chance of death by starvation or dehydration in order to protect its young, so these all inhabit the highest level of the lexicographic ordering. The remaining requirements, to perform regular scent marking, chase off intruders, avoid being far from cover, and to scan the surroundings do not dominate each other, but should always be performed in preference to grooming, and so are placed on their own level.

Note that the placing of the requirements to scan the environment and maintain proximity to cover on a lower level to that of avoiding death by starvation or dehydration does not mean that the desire to stay near cover does not influence the agent's behaviour in approaching food or water. Should an agent perceive a food source the top level requirement of avoiding death by starvation is indifferent to whether the agent

approaches directly and nonchalantly, or by staying close to cover and remaining alert. Due to this indifference, the lower level requirements to stay close to cover and scan the environment regularly dictate that the latter method is more desirable.

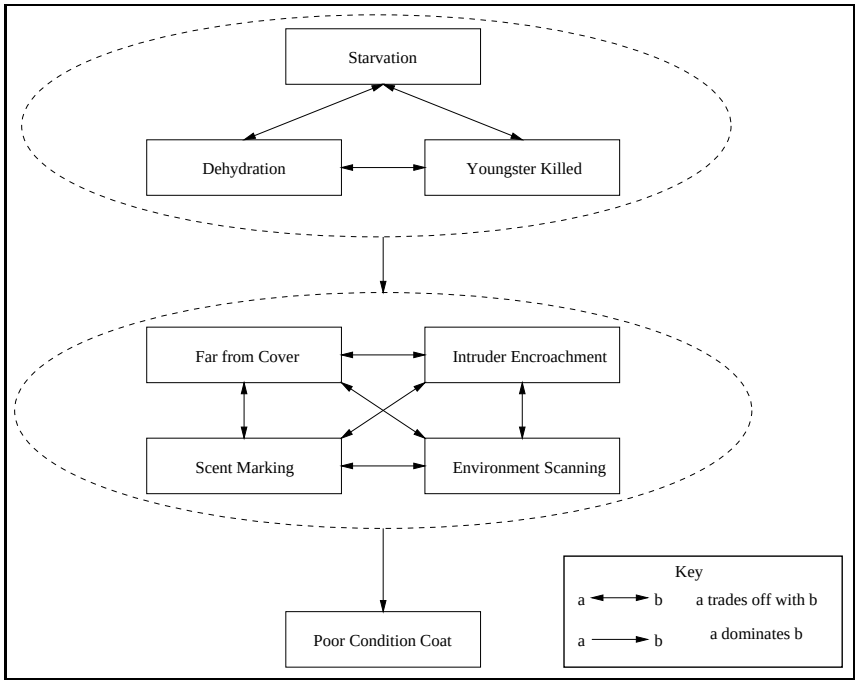

Fig. 1. General lexicographic ordering for the virtual creature.

3 Measuring Performance

An agent may not be able to satisfy all of its requirements over its lifetime, and it may suffer from many combinations of violations. In order to rate the agent's overall performance, and possible compare different agent architectures in the same environment with the same requirements we would like a measure of how well an agent has satisfied its requirements over its lifetime. We develop such a measure based on the notion of $cost$. At any instant the agent may have several violated requirements, and each violated requirement will have an instantaneous associated cost that is accrued over the agents lifetime. An ideal agent will minimize this measure over its lifetime given the resources that it has.

3.1 Violation Costs

The instantaneous cost of a violation will depend on how important the requirement is, and the nature of the violation. Some requirements may have differing levels of violation. For example, a virtual creature that is required to maintain a good coat may meet or violate this requirement at varying levels.

To distinguish between levels of violation we introduce a variable x_r for each requirement r, which measures the extent of the violation. In the case where all violations of a requirement are considered equal (such as death) the variable x_r takes the values $true$ or $false$ indicating the presence or absence of a violation.

In the case of a requirement that constrains the value of some measurable variable in the environment we can use that variable directly. In Figure 2 we present the violation cost function for the virtual creature's requirement to perform territorial marking every two days. For this requirement the instantaneous cost of violation increases with the time since violation, making it less desirable to avoid scent marking for four days in a row, than to avoid it for three days twice in succession. The exact shape of the function will depend on the chance of a competitor encroaching on the creature's territory given that it has not performed scent marking recently.

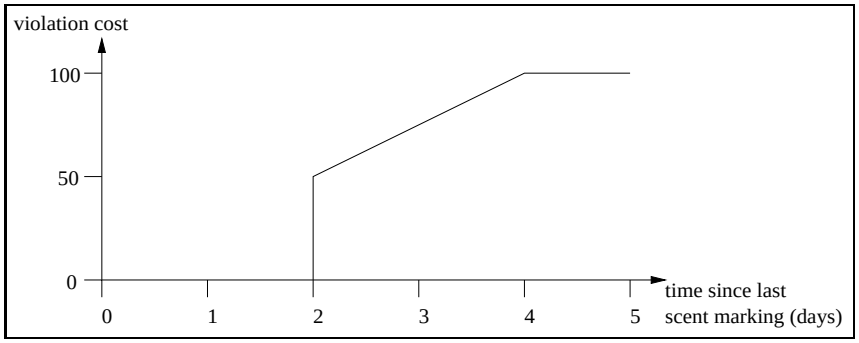

Fig. 2. Cost of violating scent marking requirement

Even in the case of requirements that are not periodic we may wish to increase the cost of a violation as time goes on. For example, we may wish to consider an agent's death 10 minutes before the required lifetime as more than twice as bad as an agent's death 5 minutes before the required lifetime.

The general form of an instantaneous violation cost function of a requirement r is a function of x_r and v_r, where v_r is the time since violation.

We can define the total instantaneous violation cost, $\mathcal{V}_l$, at each level, l, of the lexicographic preference ordering as simply the sum of the instantaneous violation costs of the requirements at that level. We obtain the overall violation cost, $\mathcal{C}_l$, at at each level is

by integrating the instantaneous violation cost with respect to time.

$$\mathcal{C}_l = \int_0^L \mathcal{V}_l(t)\ dt$$

where L is the desired lifetime of the agent. Then to compare the performance of two agents we begin by comparing the overall violation costs at each level of the lexicographic ordering in turn. We consider each level in turn, from the most important to the least important, until we find a level at which the agent's overall violation cost differs. The agent that performed best is that agent which achieved a lower overall violation cost at this level, irrespective of performance on lower levels. We disregard the agent's performance at lower levels since these are defined to be infinitely less important than the requirements at the level in which the agents are first found to differ.

In trying to minimize the overall violation cost at any level of the lexicographic ordering the agent must consider trade-offs between the requirements at that level. An agent may pursue a course of action that allows it to maintain several requirements that are individually less important than another requirement at that level, but in conjunction are more important. Developing an architecture that is capable of predicting and taking advantage of such trade-offs is the subject of the next section.

4 Motivating Autonomous Agents

Now that we have an intuitive conceptual model of how an autonomous agent should behave that is independent of the specific agent architecture employed, we can demonstrate its value by closely marrying it with equally intuitive design and implementation methods. The need for intuitive modelling techniques in agent-oriented programming has been noted elsewhere [4]. In this section we present an agent architecture specifically developed for autonomous agents with the kinds of requirements discussed in Section 2. The architecture is inspired by the distributed Multi-Agent Reasoning System (dMARS) [3], a highly successful commercial architecture based on the belief-desire-intention (BDI) model [2,6,7].

4.1 The Architecture

Our conceptual model of autonomous agent behaviour revolves around its requirements. Up to now we have discussed the requirements of an agent without reference to its capabilities. Any implemented autonomous agent will have limited resources over which its requirements will compete. In the case of the virtual creature discussed earlier the primary resource is the physical embodiment of the agent itself, which can only be in one place at any given moment. The requirement to avoid death by starvation will be best satisfied if the agent maintains close proximity to a plentiful supply of food, but at the same the requirement to avoid death by dehydration will be best satisfied if the agent stays close to a supply of fresh water. In cases where it is not possible to maintain both these conditions at the same time a choice must be made as to which requirement takes control of the agent's position. It is the reconciliation of such competition that is the main problem addressed by this architecture.

The primary components are *motivators* and *resource controllers*. A motivator serves as an intuitive encapsulation of all the information needed to satisfy a single requirement, and as that requirement's representative in the system. A resource controller is provided for each of the agent's resources that one or more motivators may want to control. When a motivator wishes to use a resource it makes a request to the appropriate resource controller. The controller passes on the details of the request to all other motivators that can be affected by that resource who are then free to criticize the proposed use of the resource. The dependencies between motivators and resource controllers are represented explicitly, so that the resource controller knows exactly which motivators can be affected by the use of its resource. We use the resource controller to arbitrate between competing motivators so that each motivator only has to understand the interface of the resource controllers that it needs or can be affected by, it does not need to know any details about other motivators in the system. Motivators and resource controllers are considered to exist in their own namespace and communicate by message passing. As such, they can be seen as agents in their own right, and the overall behaviour of the creature can be seen as governed by a multi-agent system. An example organization of motivators and resource controllers is shown in Figure 3.

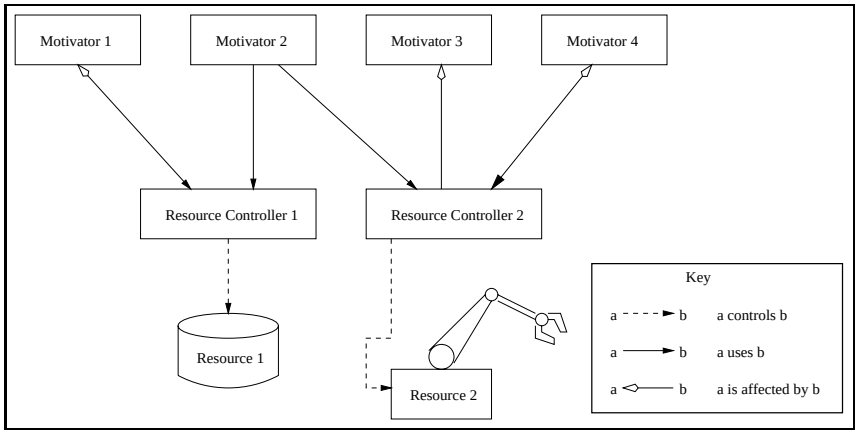

Fig. 3. Relationships between motivators and resource controllers

Individual motivators are implemented as an augmented dMARS agent as shown in Figure 4. A standard dMARS agent consists of a set of beliefs, intentions, and a plan library. Each plan in its library has a triggering condition and a context. The triggering condition determines when the plan is considered *relevant* for execution, with plans being triggered by events either received from the environment or generated internally. The plan's context determines whether a relevant plan is *applicable* for taking on as an intention, and is matched against the agent's beliefs for the plan to be considered applicable. A plan that is both relevant and applicable is taken on as an intention which

the agent is committed to act towards. More detail on the workings of dMARS agents can be found in [3].

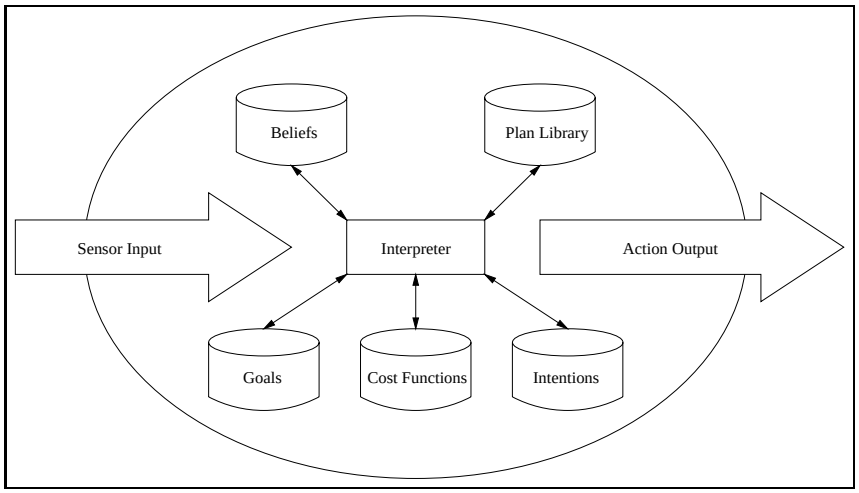

Fig. 4. An augmented dMARS agent

Motivators differ from dMARS agents in that a relevant and applicable plan is not immediately taken on as an intention, and is still only a *candidate* for execution. Should the plan involve the use of a resource that is contended between motivators, then it must request the use of that resource from its resource controller. In order to decide whether the plan is in the best interests of the agent as a whole, all of the motivators that can be affected by the plan must be allowed to criticize it. To enable this, each motivator is provided with a *benefit calculation mechanism*, which may depend on the agent's current beliefs and intentions. (Note that the calculation of benefit could be performed in a standard dMARS agent by using special plans in its plan library that are applicable when the motivator receives a request for a benefit calculation. However, we choose to seperate out the benefit calculation mechanism because it performs such a critical role in this architecture and thus the mechanism used should be readily identifiable.)

A motivator calculates the benefit of a resource use, U, using the following formula.

$$benefit = cost\ of\ not\ doing\ U\ -\ cost\ of\ doing\ U$$

We take into account the cost of not doing U to allow for situations when even the best thing we can do is expected to incur some cost, but doing nothing will incur a greater cost. An example of such a situation is given in Section 5.

Resource controllers receive proposals for use from motivators, along with an expected benefit. The details of the proposed use are passed on to all motivators that can be affected by it. These motivators return an expected benefit of the use, which may be negative if the use increases the risk of violation of the motivator's requirement.

The benefits returned by the motivators are compared on the basis of their ranking in the lexicographic ordering, starting with the greatest level of importance. The individual benefits of the motivators at each level are summed to get the overall benefit. This overall benefit is compared with the overall benefit of the current activity using the resource, or zero if there is none. If the overall benefit of the proposed use is greater than the current overall benefit at that level then the proposed use is accepted, and the current activity is suspended (and can be withdrawn altogether if the motive that generated it wishes). If it is less then the proposed use is refused, and if the benefits are equal then we consider the next level. If after considering all levels we are still indifferent between the proposed use and the current use then we choose to continue with the current use.

The resource controller maintains a record of the expected benefit of the current activity using the resource, and it is the responsibility of individual motivators to inform the resource controller should their disposition towards the current activity change.

5 Emergency Services Coordination

In Section 2 we discussed the kinds of behaviour we might require of an autonomous agent. To demonstrate the diversity of behaviours it was intuitive to discuss them in relation to a virtual creature with the same needs as a real, biological creature. However, the kinds of behaviour specified can also be applied to real-world applications where the agents have no direct similarity with biological creatures. In this section we look at one such application, that of *emergency services coordination*.

The system developed in this section is intended as an illustrative example of the potential of the architecture described in Section 4. As such, we take a simplified view of the problem of emergency services coordination, concentrating on the role of ambulances in responding to clients from the public in a simulated city. The city road layout is shown in Figure 5, and is loosely based on the layout of a city in the north-east of the United States. The hospital is the dark grey box in the north-western portion of the city.

All roads in the city are two-way, and the traffic flow is constant and uniform throughout the city. A single hospital exists in the city, which deals with all of the city's needs. Emergencies will occur at random intervals and at random points within the city, but always at a point that an ambulance can access. We refer to the object of the emergency as the *client.* Every emergency is considered severe enough to warrant transport to hospital, but the urgency of such transport may vary. In this simulation we provide three levels of urgency.

level 1 emergencies are the most urgent with the client's condition expected to deteriorate rapidly;

level 2 emergencies are urgent with the client's condition deteriorating over time, but not as rapidly as a level 1 emergency; and

level 3 emergencies are relatively stable with some deterioration over time, but nowhere near as pronounced as for a level 1 or level 2 emergency.

A number of ambulances will be available for picking up and ferrying clients to a hospital.

Fig. 5. Virtual City Layout.

5.1 Desired System Behaviour

Clearly the desired behaviour of the system is to deliver clients to the hospital promptly. We might choose to measure the overall performance as the mean time between an emergency request and delivery to the hospital. However, since we distinguish between several types of emergency, each with a differing level of urgency, we construct a more sophisticated measure. We assume that there is some measure of cost associated with the time taken to deliver a client to hospital. This measure of cost is linked to the client's chance of survival, and also perhaps the resources required to save their life. The cost functions for the three levels of emergency are given in Figure 6.

Given such a set of cost functions, the desired performance of the system as a whole is to minimize the accumulated cost over the system's lifetime.

5.2 An Agent-Oriented Approach

The primary entities within the system are the ambulances and clients. We consider both as a kind of autonomous agent, where ambulance agents and client agents will coordinate in order to ensure the prompt delivery of high urgency clients to the hospital.

A client agent's rôle is to provide the ambulance agents with up to date information concerning the urgency of the emergency, and the time the client has been waiting. The client agent receives offers from the ambulance agents for a possible pick up, along with an estimated time of arrival. The client agent selects the nearest ambulance and requests a commitment to the pick up. Since the ambulance agents are autonomous and

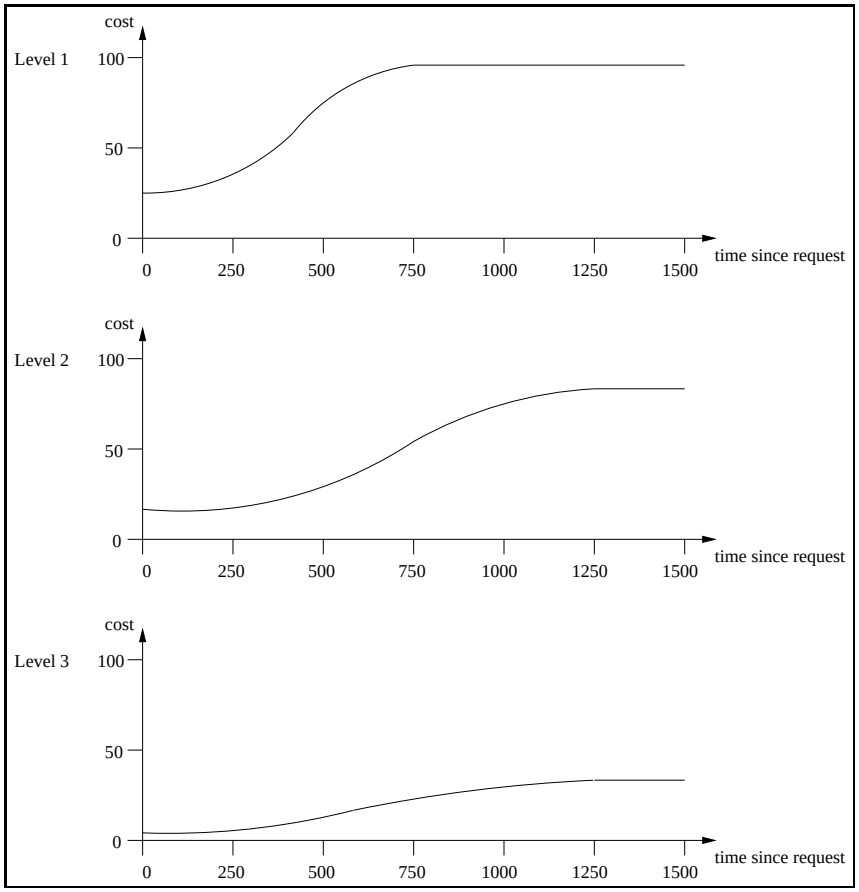

Fig. 6. Cost functions for each level of emergency

have the final say over their use, such a commitment may not be forthcoming, in which case the client agent requests a commitment from the next best ambulances until such a commitment is gained. The client agent will try to ensure that at most one ambulance is en route at any given moment.

The ambulance agents make the final choice as to which client to attend to. The decision is based on the urgency of the emergencies and the estimated time that the ambulance can get the client to the hospital. Each ambulance has a limited fuel load and must ensure that it is sufficient before committing to picking up a client. Should there be no clients to pick up the ambulances roam to different parts of the city, thereby reducing the expected journey time for any new client compared to if the ambulances remained at the hospital at such times. This roaming behaviour is only effective if the ambulance maintains enough fuel to provide full coverage of the city, otherwise the expected journey time will be increased due to the need to refuel first. Since the overall

system performance measure does not incorporate fuel costs, we are justified in using the roaming behaviour to reduce the expected delivery times and hence the overall accrued cost. In this simulation we only allow the ambulances to refuel at the hospital. In the following sections we describe the requirements of each kind of agent.

5.3 Ambulance Agents

The complete list of requirements for our ambulance agents are given below.

- Do not run out of fuel (avoidance requirement, violations are lethal).
- Pick up clients (social, one-off tasks with deadline, positive trap violation).
- Roam (default avoidance requirement, flexible violations).
- Maintain enough fuel to provide city-wide coverage (avoidance requirement, flexible violatons).

The requirements to pick up clients are social requirements that may be satisfied by any of the ambulances in the system, although at any time we will only have one ambulance committed to picking up that client. The lexicographic ordering of these requirements is given in Figure 7.

Do Not Run Out of Fuel Since the requirement to avoid running out of fuel is the sole occupoant of the top level of the lexicographic ordering we do not need to consider any trade-offs. As such, the choice of the level cost for requirement violation is arbitrary, and any cost greater than zero will suffice since behaviours defending this requirement will automatically dominate all others when a violation is predicted. In this case we set the cost of violation at 100.

In this simulation, the ambulance agents are aware of their fuel load with complete accuracy, and are also able to predict the fuel used in travelling between two points with complete accuracy. In addition, there are no possible circumstances arising that may unexpectedly alter the rate of fuel usage. Due to the lack of uncertainty in this situation; the expected cost (in relation to the fuel requirement) for any journey that terminates with the ambulance at the hospital with any positive amount of fuel left is zero. The expected cost only becomes non-zero if the ambulance is predicted to arrive at the hospital with a negative amount of fuel, i.e. it will run out on the way. This cost function is shown in Figure 8. If the measurement of fuel load or fuel usage was susceptible to error then the expected cost of a journey terminating at the hospital would increase as a function of the fuel deficit [5].

Pick up Clients A new requirement to pick up a client is instantiated every time a new emergency request is received, and is open for consideration by all of the ambulances present in the system. At any time the number of requirements to pick up clients is exactly equal to the number of clients waiting to be picked up. Each ambulance maintains beliefs about the location, emergency level, and time of request for each pending client so that they can calculate the expected costs of potential action sequences using the cost functions given in Figure 6.

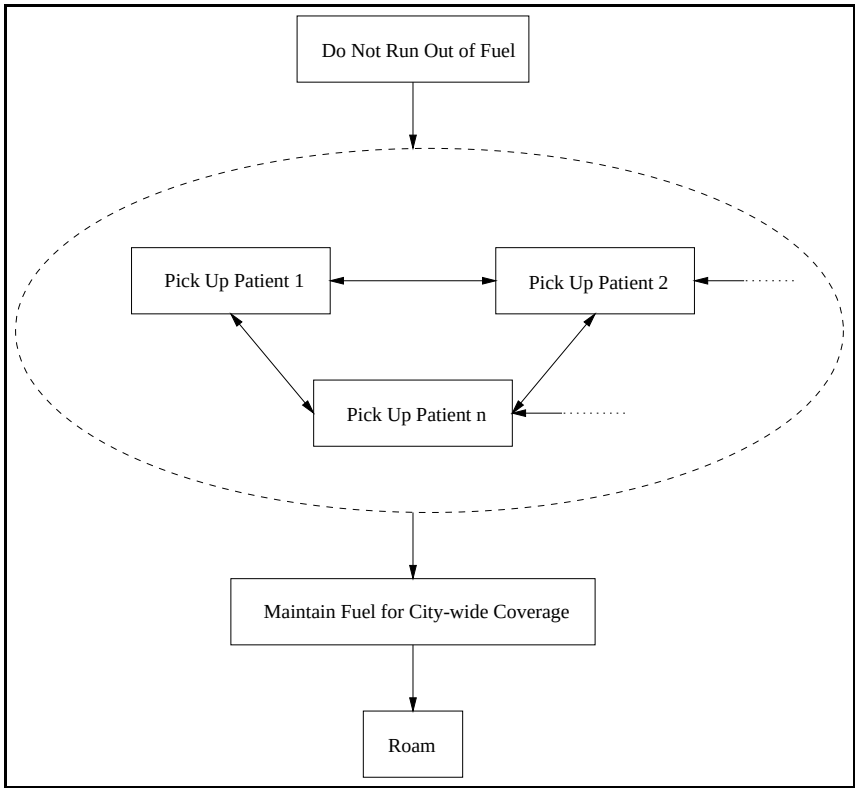

Fig. 7. Lexicographic ordering of ambulance requirements

Plans to pick up a client are considered *relevant* whenever a new emergency request is received, or whenever the ambulance becomes available after dropping off a client. When a plan to pick up a client becomes *applicable* the ambulance will send an offer to the client agent of a pickup, together with the expected journey time. These offers are regarded as prospective and without commitment. The client agent can then send a message back asking for the nearest ambulance to commit to picking it up, together with the next best offer it has received. At this stage the ambulance agent calculates the cost of picking up that client, and if the expected cost is less than the expected cost of not picking up the client then the ambulance agent commits to doing so, possibly dropping other commitments to other client agents in the process. This communication process is shown in Figure 9. How the estimated cost of these plans is calculated is outlined below.

The motivator proposing the intention to pick up a client, E_1, calculates the benefit of the intention by considering how long it will take for this ambulance to pick up the client, and how long it estimates it will take for another ambulance to pick up the client.

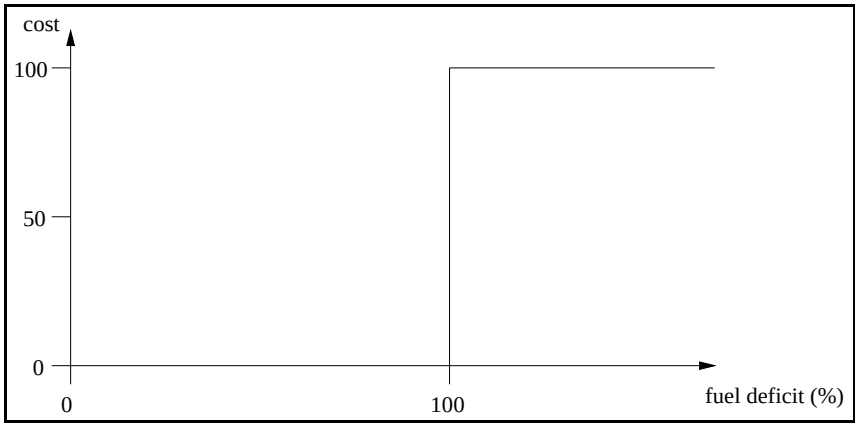

Fig. 8. Expected cost of a journey terminating at the hospital, dependent on predicted fuel deficit.

This benefit is given by the following formula. (see Section 4.1).

$$\begin{aligned} benefit &= cost\ of\ not\ picking\ up\ E_1\ -\ cost\ of\ picking\ up\ E_1 \\ &= c_{l(E_1)}(t_j(E_1) + t_{nbo}(E_1)) - \\ &\quad c_{l(E_1)}(t_j(E_1) + t_p(E_1)) \end{aligned}$$

where $c_{l(E_1)}$ is the cost function associated with the emergency level, $t_j(E_1)$ is the expected journey time from the scene of the emergency to the hospital, $t_p(E_1)$ is the estimated time of pickup if we deal with the request, and $t_{nbo}(E_1)$ is the estimated time of pickup by the next best offer that the client has received. Since $t_{nbo}(E_1)$ is greater than $t_p(E_1)$ the expected benefit will be greater than zero.

In the case where the client agent has not received another offer then we need to estimate $t_{nbo}(E_1)$, and to ensure that commitment is beneficial this estimate of $t_{nbo}(E_1)$ must be greater than $t_p(E_1)$. The client will receive another offer when another ambulance has returned to the hospital with a client, and since we know the expected journey time from the hospital to the client is $t_j(E_1)$, which will be the offer received. Given this we can estimate $t_{nbo}(E_1)$ as $t_j(E_1) + t_{interval}$, where $t_{interval}$ is the mean interval between ambulances returning to the hospital. However, in the case where the hospital is between the ambulance and the client this estimate of $t_{nbo}(E_1)$ may be less than $t_p(E_1)$. To ensure that commitment is always beneficial we take the estimate of $t_{nbo}(E_1)$ to be the maximum of $t_j(E_1) + t_{interval}$ and $t_p(E_1) + 20$. As we shall see in Section 5.4, if a client agent receives a better offer then it will release the currently committed ambulance and take on this better offer.

When considering the expected costs of picking up a client, only those motivators that are committed to a client are allowed to criticize a suggested intention to pick up another client. Those requirements that have not been committed to will already have been considered earlier in the process and intentions to pick up those clients will have been found to be inferior to those currently committed to. Thus, we do not need to

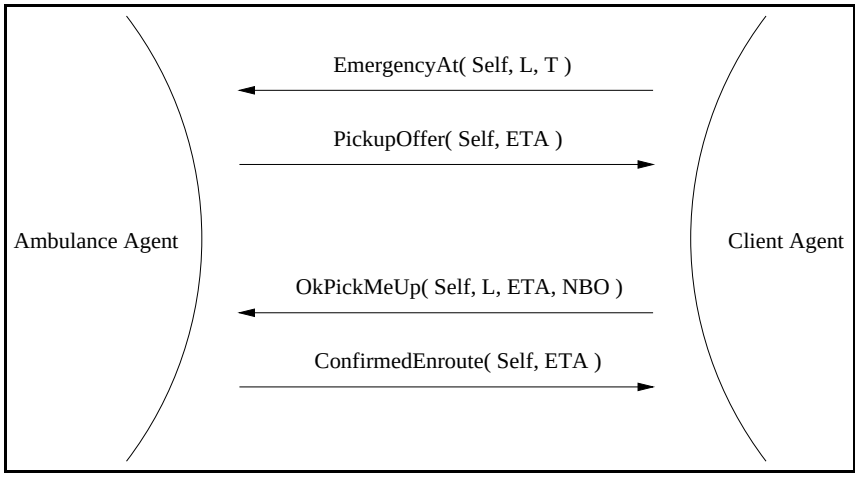

Fig. 9. Ambulance Agent – Client Agent Communication

consider every client in the system each time we consider a new intention to pick up a client.

If a motivator is committed to picking up a client, E_2, then this motivator will criticize proposed intentions to pick up another client using the following formula.

$$\begin{aligned} benefit &= cost\ of\ picking\ up\ E_2\ -\ cost\ of\ not\ picking\ up\ E_2 \\ &= c_{l(E_2)}(2 * t_j(E_2) + t_{interval}) - \\ &\quad c_{l(E_2)}(t_j(E_2) + t_p(E_2)) \end{aligned}$$

Since the commitment to E_2 will have been given some time in the past E_2 will not have an up to date next best offer. As before, the expected time for another ambulance to pick up that client is based on the interval between ambulances returning to hospital and the expected journey time between hospital and client, and back again.

Maintain Fuel for City-wide Coverage This requirement ensures that a roaming ambulance will be able to pick up a new client from anywhere within the city. It is dominated by the requirements to pick up and deliver clients should the ambulance be able to. Although the ambulance may not have enough fuel for city-wide coverage it may still be able to ferry a particular client back to the hospital.

Since this requirement is alone in its level in the lexicographic ordering, the cost we associate with violation is arbitrary, and in this case we again set it at 100. The cost function for this requirement is identical to that given in Figure 8, where we predict the amount of fuel left after a hypothetical journey taking us from our current position to the farthest point in the city, and back to the hospital.

Roam Roaming is a default behaviour that is the sole inhabitant of the bottom level of the lexicographic ordering. If no other requirement is currently directing the ambulance

then the roaming requirement will generate an intention to travel to a part of the city that is not currently well covered by other ambulances. Again, this requirement is alone in its level, so the exact cost we associate with not roaming is irrelevant. Again, we choose the value 100.

5.4 Client Agents

Each new emergency request is embodied within an agent that maintains information about the level of the emergency and the time of the request. This agent has a single requirement, to be picked up promptly.

Be Picked up Promptly On creation the client agent broadcasts the nature of the emergency and its location to all ambulances, and each ambulance that is not already carrying a client will supply a prospective offer of a pickup to this agent, supplying an estimated time of arrival. The emergency agent allows 5 seconds for offers to be received, and should any ambulance take longer than this to reply then it is likely to be involved in dealing with other clients and not the best candidate for picking this client up promptly. Should no offers be received in this time then we assume that all the ambulance are busy, and each ambulance will supply offers to all pending clients as soon as it becomes free.

As the cost of being delivered to the hospital increases monotonically with time (see Figure 6) the best offer is the one from the ambulance that will arrive first. The nearest ambulance is asked to commit to picking the client up, as shown in Figure 9. If the ambulance refuses to commit (because it has decided that it can handle another client better) then the ambulance that provided the next best offer is asked to commit, and so on until we receive a commitment. Should no ambulance commit to picking the client up then again they must all be busy serving other clients and will have to wait until one becomes free, at which time it will offer to this client agent again.

While the currently committed ambulance is en-route a better offer may be provided by another ambulance that has just dropped off a client. In this case that ambulance is asked to commit to the client agent, and if such a commitment is gained then the first ambulance is released from its commitment to this client. Once released from its commitment that ambulance will make offers to other pending clients. The communication involved in this situation is given in Figure 10.

It is possible that an ambulance currently committed to pick us up may change its mind en-route, usually because another emergency request has been received that that ambulance is better able to deal with. In this case it is possible that there are other ambulances able to respond that we have already refused offers from, so we broadcast the fact that we are again open to new offers to all ambulances.

6 Conclusion

This paper has demonstrated the beginnings of a framework for specifying, designing, and implementing autonomous agents. Using the requirements of an agent as the central

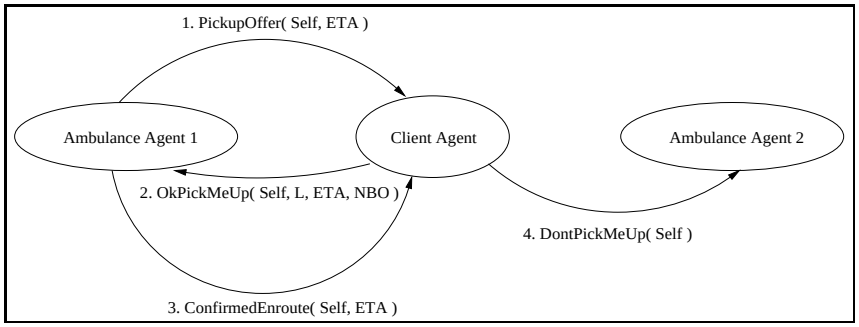

Fig. 10. Ambulance Agent – Client Agent Communication.

notion throughout means that each stage of the process remains intuitive and progresses naturally from the others.

The architecture allows the designer of an autonomous agent with limited resources and multiple requirements to specify the relative importance of its requirements, varying from strict dominance to a flexible trade-off mechanism. A major benefit of the architecture is that the relative importance of the requirements is determined by a mechanism entirely seperate from the procedural knowledge of the agent. This allows changes in the relative importance of the requirements to be performed without possibility of affecting the agents overall capabilities, and vice versa, and also enables new requirements to be added or existing ones removed without affecting any other requirements.

References

1. C. Balkenius. *Natural Intelligence in Artificial Creatures*. PhD thesis, Lund University Cognitive Studies, 1995.
2. M. E. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
3. M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In A. Singh, M. Rao and M. Wooldridge, editors, *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages*, volume 1365, pages 155–176. Springer-Verlag, 1998.
4. David Kinny and Michael Georgeff. Modelling and design of multi-agent systems. In Jörg P. Müller, Michael J Wooldridge, and Nicholas R. Jennings, editors, *Intelligent Agents III: Proceedings of the ECAI'96 Workshop on Agent Theories, Architectures, and Languages, LNAI 1193*, pages 1–20. Springer-Verlag, 1996.
5. D. McFarland and T. Bösser. *Intelligent Behaviour in Animals and Robots*. MIT Press, Cambridge, MA, 1993.
6. Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a bdi-architecture. Technical Report 14, Australian Artificial Intelligence Institute, February 1991.
7. Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

8. Stuart Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, 1995.
9. Toby Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, 1993.