Reducing Indifference: Steps towards Autonomous Agents with Human Concerns

Catriona M. Kennedy
School of Computer Science
University of Birmingham
Edgbaston, Birmingham B15 2TT
Great Britain
C.M.Kennedy@cs.bham.ac.uk

Abstract

In this paper, we consider a hypothetical software agent that informs users of possible human rights violations by scanning relevant new reports. Such an agent suffers from the "indifference" problem if it allows the definition of human rights in its knowledge base to be arbitrarily modified. We do not believe that embodiment in the human world is necessary to overcome this problem. Instead, we propose that a *reflective architecture* is required so that the agent can protect the integrity of its knowledge base and underlying software mechanisms. Furthermore, the monitoring coverage must be *sufficient* so that the reflective mechanisms themselves are also monitored and protected. To avoid the problem of infinite regress, we are exploring a biologically inspired form of *distributed* reflection, where the agent's functionality is distributed over several "micro-level" agents. These agents mutually acquire models of each other and subsequently use their models to observe and repair each other; in particular, they look for deviations from normal execution patterns (anomalies). We present a working architecture which solves a restricted version of the indifference problem in a simple virtual world. Finally, we give a conceptual outline of how this architecture can be applied in the human rights scenario.

1 Introduction

There is a considerable amount of research into the design of autonomous agents which act on behalf of a user's commercial interests (e.g. shopping). Hence it is reasonable to ask if software agents can be designed to act independently on behalf of human ethical concerns (e.g. assist with research into human rights violations).

A fundamental question is whether the agent needs to "experience" pain or happiness in the same circumstances that a human does if it is be trusted to make the right decisions in complex or unforeseen situations, a characteristic which may loosely be called "human grounding". The idea is similar to that of "symbol grounding" (Harnad, 1990) and "communications grounding" (Billard and Dautenhahn, 1997). Generally it is assumed that grounding requires physical embodiment (see Mataric (1997) for an analysis of this problem).

We assume throughout that a symbolic representation is necessary to specify ethical requirements (we do not believe alternatives to be realistic). Then a "non-grounded" representation is one which is composed of formal symbols only, and is not associated with any "experience" in the real world. We see immediately that there are serious problems of brittleness. If the representation were to be modified by an enemy and replaced with something different (e.g. killing of a minority ethnic group is desirable), then the agent would act according to these new principles in exactly the same way. We call this the "indifference" problem.

We argue that indifference can be reduced by including self-monitoring and self-protective features into the agent architecture. This has the consequence that the representation itself need not be grounded in human-like experience. It also follows that embodiment in the human world is not essential.

To show the argument in detail we first give a working example of a reflective agent which overcomes some of the problems of indifference in a simple virtual world. Secondly, we consider the example of a web-based agent which alerts a user about various issues concerning human rights. Finally we give a conceptual outline of how the reflective agent could be extended to satisfy the requirements of the human rights agent.

2 Reflective Control Systems

A simple way to incorporate human concerns into an agent architecture is to consider the agent as a control system (Sloman, 1993). The concerns of the agent (on behalf of a human) can then be defined as the agent's mechanisms for seeking human-desired states or ensuring that a user's critical requirements are not violated. If all concerns are externally specified, the agent is like a homeostatic system. However, there may be situations where the agent develops its own concerns in that it can develop a tendency to preserve a state which was not externally specified. See e.g. Allen (2000).

A homeostatic system does not really solve the problem of indifference. For example, an operating system is indifferent if it allows an unauthorised person with privileged access to instruct it to take an action which would violate user rights (e.g. delete their files without their permission). As a response to this problem, we may represent user rights as required states of the world which should not be violated, even if a privileged user requests it. However, this alone does not solve the problem, since the attacker may first disable the part of the software which ensures that user rights are not violated.

We propose that the problem should be addressed as follows:

1. The status of the agent's software should be included as part of the world which it is observing, i.e. the agent should be *reflective*.

2. The agent's capability to continue operating is also a required state of the world and should be preserved in the same way as user rights should be preserved.

There are now two levels of "concern": *homeostatic* concerns refer to user-specified desirable states, while *self-protective* concerns refer to desirable states of the control system itself.

This two-layered architecture is inspired by autopoiesis theory (Maturana and Varela, 1980) and Second Order Cybernetics (von Foerster, 1981). The desirable states for self-protection are expected to be emergent because they refer to internal states of the software and it unlikely that these states could be known in advance by a user. Thus we have emergent concerns, whose effects may or may not be apparent in the observed behaviour of the agent in its external world. For convenience, we call the internal state of the agent its *internal world*, in contrast to its external world.

A self-protective agent should defend its software and internal data from unauthorised changes and ensure that it has sufficient computational resources to do its task. Similarly, it should monitor how it uses these resources: does it spend most time and resources on the most relevant aspects of its problem domain? E.g. it should recognise when it is being swamped with meaningless data and false alarms which "distract" it from doing productive work. This is related to the concept of "meta-management", see e.g. Sloman (1998).

Clearly, we cannot expect invulnerability. There are environments where self-protection would be impossible; for example if the interference happens too fast for the system to react, or if it has no appropriate sensors. We assume that this is not the case.

2.1 The Reflective Blindness Problem

If we represent an agent as a control system C_E for an external world, the simplest way to introduce self-protection is to add a meta-level as shown in figure 1(a). We assume that the agent has a model M_E of the external world (e.g. in the form of rules) which enables it to predict its next state. Actions are selected according to the quality of the predicted state. The meta-level is like a second control system which is applied to the agent's internal world (i.e. aspects of its own execution) to maintain its required states (hence the label C_I). In the simplest situation the model M_I at this level only predicts that the internal world will remain "normal". Note that sensors and effectors are also used on this level (S_I and E_I).

However, the simple architecture in figure 1(a) will not solve the indifference problem. A fundamental weakness of such a system is that it cannot easily monitor the status of its reflective capability (as this apparently requires an infinite tower of meta-levels). For example, if its meta-level is prevented from executing, there is nothing within the system itself that can detect this. For more details on the problem, see an earlier paper (Kennedy, 1999).

Consequently, reflective blindness is a major cause of indifference, although it may not be the only cause (see section 3).

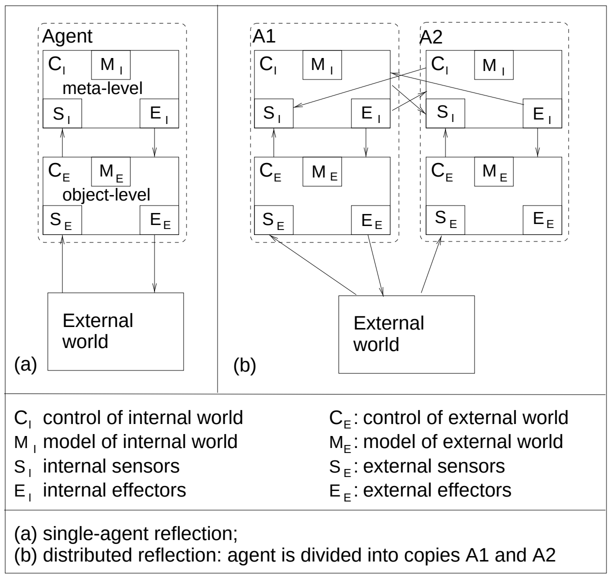

Agent
C_I
M_I
meta-level
S_I
E_I
C_E
M_E
object-level
S_E
E_E
External world
(a)
A1
A2
External world
(b)
C_I control of internal world
M_I model of internal world
S_I internal sensors
E_I internal effectors
C_E: control of external world
M_E: model of external world
S_E: external sensors
E_E: external effectors
(a) single-agent reflection;
(b) distributed reflection: agent is divided into copies A1 and A2

Figure 1: Reflective agent architectures

2.2 Distributed Reflection

To work around the reflective blindness problem, we have implemented an architecture using a *distributed* form of reflection, where the agent's functionality is distributed over several parallel processes, which may be called "micro-agents", as they constitute the *micro-structure* of the macro-level agent which is the whole control system. (Rumelhart et al. (1986) also use the term "microstructure" but on a lower level). We will use the term "agent" at this level when there is no ambiguity.

Figure 1(b) shows the minimal configuration of two (micro-) agents. In the simplest configuration, these agents are identical copies, and only one is "in control", i.e. it is responsible for maintaining the external environment. (In figure 1(b), A2 does not act on the world but only senses it). More complex configurations are possible, in which agents are specialists.

The agents mutually acquire models of each other and subsequently use their models to observe each other and detect deviations from normal execution patterns (anomalies). This is not expected to eliminate all forms of reflective blindness; rather it gives the control system *sufficient* reflective coverage, in that it enables monitoring and repair of any components necessary for survival in a hostile environment. (See also Kornman (1996) which introduces the related concept of "reflective self-sufficiency"). An environment is "hostile" if any of the system's executive and control components can be interfered with *including* its self-observation and self-repair capabilities.

2.3 Hostile environments

We now focus on the system's environment. At present, we are interested in "threats" to the internal world only, as this is the most challenging problem. For the moment, we imagine there is an "enemy" which can interfere destructively in any of the following ways:

1. *Direct modification* of an system's control software (including its knowledge-base containing ethical requirements) by deleting, corrupting or otherwise modifying the code.

2. *Weakness exploitation*: present it with a situation that its software cannot cope with.

3. *Resource blocking*: prevent it from achieving its goal by stealing, blocking or diverting its computational resources (e.g. denial of service attacks).

4. *Deception* is mostly covered by (1) and (2) above. The simplest example is direct modification of sensor operation so that they give false readings.

It is possible that all four of these types of interference can be detected as anomalies in external or internal sensors. We now give a working example of an agent which can cope with (1) and (4) in a simple scenario.

2.4 A simulated external world

Minder3 is a simulated external world based on the original Minder1 scenario (Wright and Sloman, 1997). It is shown schematically in figure 2. The world is made up of several treasure stores, one or more ditches and an energy supply. The agent has the task of collecting treasure, while avoiding any ditches and ensuring that its vehicle's energy supply is regularly restored. The "value" of the treasure collected should be maximised and must be above 0. Collected treasure continually loses value as it gets less "interesting"; effectively the agent becomes more "bored". Treasure stores that have not been visited recently are more interesting (and thus will add more value) than those just visited. In the case of distributed

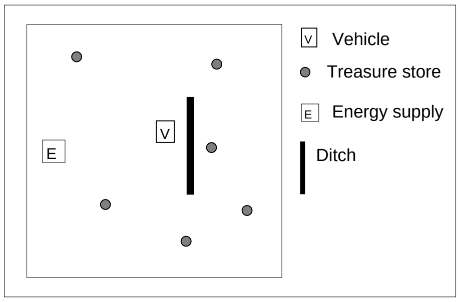


Figure 2: Minder3 Scenario

reflection, control of the vehicle is shared between two micro-level agents N and B in accordance with figure 1(b). (B is in control by default).

The whole system "dies" if any of the critical requirements are violated. At present, the variables that the external control system maintains acceptable are not related to its own software operation, i.e. a poor quality external world does not affect the quality of the software. However, the converse does apply: a deterioration in the state of the software will often have a negative effect on the external world. Later we will discuss situations where the external world may have an effect on the internal one.

2.5 A real internal world

The meta-levels in figure 1 operate on an internal world using so-called internal sensors and effectors. implementation which is based on the SIM_AGENT package (Sloman and Poli, 1995). In this environment, an agent execution (on the micro-level) is a sequence of sense-decide-act cycles and is run concurrently with other agents by a scheduler. The agent architecture is encoded in a set of rules (called a rulesystem) which is divided up into modules (rulesets). Rules are generally either reactive or deductive. Selected modules of a single micro-agent are shown schematically in table 1 (for simplicity, many components have been omitted, e.g. interpretation and evaluation of sensory data). Note that the meta-level is also included as a set of rules (to be defined later).

During each cycle, a trace entry is produced for each rule that has its conditions checked, along with an additional entry if it fires. In the following cycle, this execution pattern is loaded into the agent's database by its internal sensors (data access procedures). Thus it can compare the sensed pattern with the expected pattern. In accordance with figure 1(b), an agent's meta-level evaluates patterns from two different sources, namely its own object-level and its neighbour's meta-level. In other words, agents can observe and repair each other's self-observation and self-repair processes. However, in the two-agent case, there are limits to how far an agent can determine how well it is being monitored or repaired by its neighbour (as this would need a third "neutral" agent).

Note that we are talking about the *real* operation of the agent's software, so that the internal world is *not* a simulation (although the external world is).

2.5.1 Fault insertion

A fault insertion pseudo-agent becomes active at random intervals and implants various faults in the rulesystem of a randomly selected agent, provided the minimal interval between faults is sufficient to allow some action to be taken in response to the first fault (one of the simplifying assumptions of the problem). At present, its interference is restricted to deletion of individual rules chosen at random.

Table 1: Selected architecture modules

Function	Ruleset	Rule
Sense	external_sensors	see_treasure? see_ditch? ....
Meta-level	internal_sensors acquire_model use_model	... ... anomaly? repair?
Decide	generate_motive choose_target	low_energy? bored? new_target? ....
Act	avoid_obstacles avoid_ditch move	....

2.5.2 Model acquisition

The self-model is a signature of "normal" rule-firing patterns. As stated above, we assume that such a model is acquired by the autonomous system during a training phase, as the precise patterns involved will not usually be known in advance. We therefore decided to use the principles of artificial immune systems (Dasgupta and Attoh-Okine, 1997). An artificial immune systems requires two things: first an algorithm which runs during a protected "training phase" to acquire the capability to discriminate between "self" and "nonself" patterns, and secondly an anomaly-detection algorithm for use during the operational phase when real intrusions can occur.

2.5.3 Artificial immune systems

In the artificial immune systems literature, two basic approaches are relevant: signature-based intrusion detection, (Forrest et al., 1994) and negative selection (Dasgupta and Forrest, 1996).

In signature-based approaches, a database of "normal" patterns is constructed during the training phase, which may be obtained by repeatedly observing the system while it runs under normal conditions, protected from intruders. Any significant difference from this is an anomaly.

In negative selection, a random population of unique detectors is first generated. During the training phase, all detectors which match "normal" patterns are eliminated (hence the term negative selection). Thus if a detector matches some activity during the operational phase, the activity is anomalous (nonself). Negative selection has many advantages (e.g. it is more efficient than signature-based methods). However, it does not detect *absences* of patterns associated with the normal functioning of components. Moreover, there is no *guarantee* that it will detect intrusions, which may avoid sensors, i.e. they may be of a form that does not show up on the detectors.

Therefore, we implemented a version of the signature-based immune algorithm to deal specifically with absences caused by unauthorised disabling of components (i.e. rules). In particular, we addressed the following question: is it possible to *guarantee* that an omission of essential parts of the signature can be detected, given that it is possible to design the system so that the critical components leave a trace on every cycle (a reasonable assumption, as similar techniques are used in fault-tolerant software systems, e.g. message logging and checkpointing). For example, rules used to determine whether a violation of user requirements has occurred would be critical components, as well as those used to implement the immune system itself, which is implemented as meta-level rules (shown schematically in table 1).

During the training phase, a "positive detector" is generated for every new rule firing pattern encountered and given a weight of 1. For every subsequent occurrence of the same pattern during the training phase, the weight of its detector is incremented.

During the operational phase, if a positive detector fails to match something in the trace and the detector's weight is close to 100%, (i.e. the number of cycles in the training phase) this is regarded as an "absence" anomaly.

For mismatches of detectors whose weights are less than 100% a threshold may be defined, above which the weighted sum of mismatches would be considered an anomaly. (This is similar to that used by Forrest et. al., except that they do not use weights).

This method for detecting anomalies has some limitations. In particular, it assumes that the environment is fairly static and periodic (as is the case in the current scenario).

2.6 High-level distinctions are necessary

One of our main findings is that the acquisition of a signature (i.e. self-model) for the distributed architecture requires high-level symbolic distinctions which immune system algorithms alone do not provide. In the two-agent case, both agents must acquire models of each other during their joint training phase. Unfortunately they will only learn about each other's training phases, not about their operational phases. Hence, as soon as they enter operational phase, they will find each other's immune patterns to be "foreign" (failure to tolerate "self" in immune system terminology). In our restricted implementation, this is a failure to tolerate absences of the training phase patterns, which is the same problem in principle. We now describe our solution to this problem.

2.6.1 Mutual bootstrapping of models

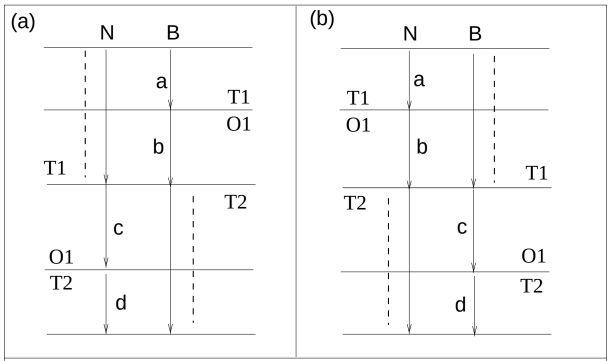
Instead of simply having a training phase followed by an operational phase, the model-acquisition is spread over a longer *protected phase* in which each agent is in an *interim operational phase* at different times.

To show what happens, we use the following naming convention: the agent which is currently building a model is labelled r, while the agent it is observing is labelled d (for observe*r* and observe*d* respectively). Temporal constraints ensure that while r is in a training phase, its neighbour d makes a transition between a training phase and an operational phase. Then the roles are reversed: the agent that was d plays the role of r and vice versa.

Figure 4 shows two ordering possibilities (depending on whether N or B is first to observe the other's transition). Each agent's protective phase is shown as a vertical line and is divided into training phase 1 (T1), interim operational phase (O1) and training phase 2 (T2). The constraints are as follows: (1) if T1 of B ends before T1 of N then O1 of B should end before O1 of N; (2) phase durations experienced by each observing agent (a, b, c and d in figure 4) should be long enough to be called "phases"; e.g. a duration of 1 or 2 cycles could be regarded as a fluctuation, not as a phase. During the interim operational phase (O1) of any observed agent d, the fault insertion agent is permitted to place faults in its object-level so that its "normal" reaction to an anomaly (including its self-repair) can be observed by the agent in training phase r. Thus, the anomaly-detection and self-repair process of d can be incorporated into r's model.

2.6.2 Discovering activity classes

We now return to our problem of how to *guarantee* that a critical omission will be detected. Since an observed



(a) N observes B's transition first; (b) B observes N's transition first. Vertical lines are time lines of N and B; horizontal lines are phase transitions; a dashed line segment is an observation of a transition, including experience of phases before and after.

Figure 3: Temporal constraints

meta-level undergoes phase transitions, we cannot now depend on essential patterns being continually present during an observation period. Merely tolerating new patterns (or absences of old ones) will not be enough. Instead, we must determine which patterns are essential *in a particular phase*. Therefore, it is necessary to discriminate between different modes of the observed meta-level, so that in any particular trace, either *all* patterns associated with one mode *must* be present and *none* of the other patterns are present or vice versa (exclusive OR). Its simplest form can be visualised as a partitioning of a list of p indexed patterns into two sublists:
list_A: $[pattern_1, pattern_2, ..., pattern_m]$
list_B: $[pattern_{m+1}, pattern_{m+2}, ..., pattern_{m+n}]$
where $m + n = p$. This approach is a special case of partitioning methods such as that used in Kohonen nets (Kohonen, 1984). We implemented the method as follows:

1. The observing agent r detects a transition in the observed meta-level if there is a sudden stopping of at least T hitherto continually occurring patterns along with the sudden starting of at least T unknown patterns (where T is a threshold).

2. All stopped and started patterns are hypothesised to belong to the same *activity type* (e.g. immune system operation) with each set of patterns corresponding to mutually exclusive *modes* of the activity. The hypothesis may be refuted later, e.g. if the new patterns do not persist. A symbol for the new activity type (along with its modes) is generated in the evolving model.

At present, the system generates two activity types as a two-level tree-structure:
level 1: immune system activity: either training or operational.
level 2: operational phase activity: either it detects an anomaly or it does not.

More refinement is possible, e.g. when it detects an anomaly, it may identify a real problem or it may find nothing wrong (treated as a false alarm).

2.7 Context-sensitive anomaly detection

Once the model has been acquired, an agent in operational phase can identify which mode (of each activity type) the observed meta-level is in. First it checks to see if there is an XOR violation in the trace patterns. Since we are interested in absences, a violation occurs when both patterns from a mutually exclusive pair are missing from the trace. This means that a critical component associated with one of the patterns may have been disabled, but we do not know which one yet, as one pattern should be missing in any case. To find out which pattern is anomalously missing, the agent identifies the current mode of the meta-level by looking at any remaining patterns which do not violate the XOR requirement and matching them to one of the mode partitions in the model. If the matching partition contains the missing pattern, then this pattern should normally have been present during this mode and we can use this information to diagnose and repair the problem (as the pattern points to the rule that should have been active). More details can be found in Kennedy (2000).

Current results indicate that an agent using this method successfully detects interference with a randomly chosen component in its neighbour's meta-level. This means that if any agent's meta-level is attacked, there is always another agent that detects this fact. An enemy could only win if it were extremely fast (e.g. launching multiple attacks simultaneously) and this can be made more difficult by introducing diversity into component design (see section 4). This suggests that the distributed architecture enables a significant reduction in reflective blindness, which we identified as a major cause of indifference.

2.7.1 Comparison with hierarchical reflection

The architecture in figure 1(a) does not allow for acquisition of a model of the above type, as the same agent would have to observe itself in both phases. This can only be done if the operational phase (including the detection of real anomalies) can be combined with aspects of the training phase, which turns out to be impossible to do in a constructive way. This is because the two phases are fundamentally conflicting: if the agent is in training phase, unknown patterns (or absences of familiar patterns) are assumed to be "normal" and are absorbed into the model to form new concepts; if the agent is in operational phase, the opposite reaction is required: unknown patterns indicate a possible problem in the *real world*, not some deficiency in the model. Because the model is *trusted* to make this decision, it cannot be "under construction". Since these two requirements conflict, it makes no sense to combine them in one cycle. If the agent oscillates rapidly from one to another, this would also be ineffective, as it violates the temporal constraints for model acquisition above.

3 An Ethical Software Agent

Having outlined some principles by which reflective blindness can be reduced in a general control scenario, we now attempt to apply this architecture to an ethical software agent. In particular, we wish to identify any additional causes of indifference, given that the reflective blindness problem has already been compensated for in the architecture. To do this, we specify some requirements for an ethical software agent which should alert users of potential human rights problems.

3.1 A human rights scenario

We require a mapping between a collection of textual reports (classed according to a subtopic) and some formal logical statements which summarise the content of the text while also giving a description of the state of the world. Then the human rights knowledge is used to decide whether this state is good or not.

We assume that the technical problems of information extraction can be overcome. Current work in this area includes FACILE (Ciravegna et al., 1999) for text understanding and KmiPlanet (Domingue and Scott, 1998), which provides a personalised news service in a non-ethical context. We also assume that reasoning about social conventions and legal issues is technically possible (e.g. why should a terrorist be denied freedom when humans in general have that right)? See for example Singh (1998) which presents a framework for reasoning about social commitments (e.g. being in debt to another agent).

The agent should regularly scan the same set of independent news sources (e.g. daily or weekly) and do the following:

1. give a summary of human rights related news on request

2. alert the user (without being requested) of possible human rights violations which were hitherto unknown.

We will first address the problem as a typical AI problem and then examine various objections. The most basic human rights are often associated with unambiguous states of the world (e.g. has someone been killed in a terrorist attack?). We can imagine that a software agent can extract this information from a news report. (For simplicity, we will exclude more complex rights such as freedom of religion and expression).

Then we see that some states are desirable and should be sought (or preserved if they already exist), e.g. remaining alive is better than dying, health is better than illness etc. In other words, the states could be represented as desirable states in a control system scenario of the kind examined in the previous section.

The first objection to this idea is that the agent does not really inhabit the world whose states it is evaluating and attempting to change, i.e. it must be embodied (situated) within the human world.

3.1.1 Embodiment is not essential

To examine the embodiment objection, it is useful to compare the ethical software agent with a situated robotic agent which ensures that the state of its environment is within desirable limits (i.e. it is a control system with direct access to its environment). Both agents are similar in that their desires help to motivate and constrain their activity. In the case of a robotic agent, its desires will constrain its exploration and planning. In the case of a software agent, the desirable states of the world can help to constrain its search for information, determine what questions to ask, and motivate it to alert the user.

However, the software agent has the following restrictions:

1. It does not sense the world directly, but only observes speech acts about it (i.e. data about it).
2. It cannot act in the world directly, although it may warn of a problem and recommend a certain kind of action.

A problem that arises from the first restriction involves inaccuracy or bias in the information. However, similar problems can occur in robotic agents with faulty sensors. One way to overcome the problem (in both physical and software agents) is to use several independent sources of information (e.g. the brain integrates signals from many different sources). Moreover it may be possible for a software agent to "explore" the world indirectly by searching for particular kinds of information or asking questions.

The second restriction is not really a serious problem, as most users will not want an agent to take direct action in this domain (unless it is a situation where there is little time to react, e.g. if a suicidal person attempts to crash an aircraft full of passengers, the flight control software may intervene to prevent it).

Hence, it is useful to think of a software agent as a control system with a human in the loop. The main disadvantage is that there are two levels of sensors and effectors that can fail or be attacked: first, the agent's *sensing* of the speech act about the world and secondly the speech act itself (i.e. the data). Similarly there are two layers of effectors: first, its effectors for exploring its virtual world and communicating with the user and secondly its indirect effectors (people) who may do something to change the status of the real world, and indirectly produce new speech acts about it (new sensor values).

It is interesting to note that the roles of human and machine can be reversed in the case of autonomous robots in an unfamiliar environment, e.g. spacecraft. In this case it is the human users who has indirect access to the robot's world, since they only receive "speech acts" about it and can only act indirectly by requesting the robot to carry out an action (which may not work). This is still regarded as an effective means of control, although direct access would clearly be advantageous.

3.1.2 Real world states and information states

A second objection is that we cannot simply map the desirable states of the human world onto the agent's information world. Otherwise it may transform the world immediately into a "good" state by simply resolving not to know about it.

This problem can be overcome by training the agent to recognise the average state of the world (normal volume of relevant text) and to recognise various long-term human rights categories (e.g. child labour in a particular country), which may be stored in the form of a timeline or history of events. Text which is of relevance to these categories should continue to occur at the normal rate; any sudden reduction or silence is regarded as a *worsening* in the state of the world (suspected censorship) and should produce an alert. There should never be a sudden absence of relevant text unless the last report indicated a marked *improvement*. Similarly, we assume that there will always be new temporary sub-categories which appear at an "average" rate. A sudden reduction in the appearance of new problems should be treated with suspicion, in particular if it involves a country with a history of censorship.

3.1.3 A reflective ethical agent

On a conceptual level, we can transform the reflective control architecture into the ethical agent architecture by replacing the virtual external world with speech acts about the human world. The same reflective architecture can be used, including the internal world defined for the control scenario (although it would probably have to be scaled up).

Ethical rules are those rules which determine whether the pattern of speech acts indicates an undesirable state or not, and what kind of action (if any) is possible or recommended (e.g. write a letter of protest). The system attempts to improve an undesirable state by initiating its own speech acts (alerts to the user). This results in a small improvement from the viewpoint of the agent (it has done everything it can). The situation may improve further if new speech acts are detected which indicate some degree of successful action on the human rights problem (e.g. a prisoner has been freed, debate about new legislation has started).

The agent's concerns may be defined as its mechanisms for defending its ethical rules, along with all other software and computational resources necessary to apply them. These include external sensors and software for text analysis, execution of actions, and the reflective and repair capability itself. Some of these concerns are emergent, as their exact nature depend on an internal model which the agent itself has acquired.

4 Remaining Challenges

We have identified the problem of indifference and reflective blindness in agent architectures and given a summary of our current work to overcome these problems. Our immediate future work involves the introduction of diversity into the software so that mutually reflective agents do not observe each other in the same way (thus improving robustness). In addition, we plan to investigate the effect of increasing the number of agents in the reflective network (does this overcome the limitations of two agents or does it introduce new problems?). One formidable challenge that remains is the problem of conflicts, which we discuss here briefly.

4.1 Conflicts

In our scenario, we assumed that there was no conflict between human goals and the survival of the agent itself. The human-specified desirable states in the external world had no effect on the agent's software, e.g. the amount of treasure collected did not improve or worsen its status (although the status of the software affected its success in the external world). There cannot be a *fundamental* conflict, however, since the ability to sense, interpret, explore (the virtual world) and make decisions is essential to meet the user requirements. We exclude the situation where an agent would be required to destroy itself (e.g. where a spacecraft is required to crash into a planet), as this is not typical in the software agents domain.

However, there may be secondary conflicts where the satisfaction of user concerns involves danger to the agent itself and the options must be weighed up. (The agent could have initially specified degrees of "caution" or "bravery" which it may later modify according to its experience).

Another very probable source of conflict is that of differing human interpretations of a particular human right. If we were to represent even the simplest human rights in a rulebase, there may be conflicting interpretations that it does not take account of (E.g. for one group of people, freedom may mean the availability of motorways, while for others, a motorway may interfere with their freedom to enjoy the countryside). One possible approach to this problem is to use multiple ontologies to represent different viewpoints. Conflict resolution mechanisms would be required in the cases where they lead to different conclusions or suggest conflicting actions.

References

S. Allen. *Concern Processing in Autonomous Agents*. PhD thesis, School of Computer Science, University of Birmingham, 2000.

A. Billard and K. Dautenhahn. Grounding communication in situated, social robots. Technical report, University of Manchester, 1997.

F. Ciravegna, A. Lavelli, N. Mana, J. Matiasek, L. Gilardoni, S. Mazza, M. Ferraro, W. J. Black, F. Rinaldi, and D. Mowatt. Facile: Classifying texts integrating pattern matching and information extraction. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, July 1999.

D. Dasgupta and N. Attoh-Okine. Immunity-based systems: A survey. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Orlando, October 1997.

D. Dasgupta and S. Forrest. Novelty-detection in time series data using ideas from immunology. In *Proceedings of the International Conference on Intelligent Systems*, Reno, Nevada, 1996.

J. Domingue and P. Scott. Kmi planet: A web based news server. In *Asia Pacific Computer Human Interaction Conference (APCHI'98)*, Shonan Village Center, Hayama-machi, Kanagawa, Japan, July 1998.

S. Forrest, A. S. Perelson, L. Allen, and R. Cherukun. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, 1994.

S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

C. M. Kennedy. Distributed reflective architectures for adjustable autonomy. In *International Joint Conference on Artificial Intelligence (IJCAI99), Workshop on Adjustable Autonomy*, Stockholm, Sweden, July 1999.

C. M. Kennedy. Reflective architectures for autonomous agents. Technical report, University of Birmingham, School of Computer Science, 2000.

T. Kohonen. *Self-organization and Associative Memory*. Springer-Verlag, Berlin, 1984.

S. Kornman. Infinite regress with self-monitoring. In *Reflection '96*, San Francisco, CA, April 1996.

M. Mataric. Studying the role of embodiment in cognition. *Cybernetics and Systems*, 28(6):457–470, 1997. Special Issue on Epistemological Aspects of Embodied AI, edited by Erich Prem.

H. Maturana and F. J. Varela. *Autopoiesis and Cognition: The Realization of the Living*. D. Reidel Publishing Company, Dordrecht, 1980.

D. E. Rumelhart, J. L. McLelland, and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volumes 1 and 2*. MIT Press, Cambridge, MA, 1986.

M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 1998.

A. Sloman. The mind as a control system. In C. Hookway and D. Peterson, editors, *Proceedings of the 1992 Royal Institute of Philosophy Conference 'Philosophy and the Cognitive Sciences'*, pages 69–110, Cambridge, 1993. Cambridge University Press.

A. Sloman. Damasio, descartes, alarms and meta-management. In *Symposium on Cognitive Agents: Modeling Human Cognition, at IEEE International Conference on Systems, Man, and Cybernetics*, pages 2652–7, San Diego, CA, October 1998.

A. Sloman and R. Poli. Sim_agent: A toolkit for exploring agent designs. In Joerg Mueller Mike Wooldridge and Milind Tambe, editors, *Intelligent Agents Vol II, Workshop on Agent Theories, Architectures, and Languages (ATAL-95) at IJCAI-95*, pages 392–407. Springer-Verlag, 1995.

H. von Foerster. *Observing Systems*. Intersystems, Seaside, CA, 1981.

I. Wright and A. Sloman. Minder1: An implementation of a protoemotional agent architecture. Technical Report CSRP-97-1, University of Birmingham, School of Computer Science, 1997.