Distributed Meta-Management for Self-Protection and Self-Explanation

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

Abstract

A non-hierarchical system in which distributed meta-levels mutually monitor and repair each other can act collectively as a self-protective system. Such a system can resist attacks against any part of itself, including its self-monitoring and self-repair processes. A distributed self-model is acquired by a process of bootstrapping where the different components learn by mutually observing each others' internal processes. In this paper, we argue that such a decentralised architecture is also capable of high level reasoning and self-explanation. Using some examples, we show how objectively defined interactions between distributed software components can be translated into a subjective narrative about the agent's own mental states. Furthermore, the global coordination needed to provide coherent actions and explanations is possible in a decentralised architecture, provided that the meta-level which is managing the coordination is also subjected to critical monitoring and modification. An important principle is that the different meta-levels are not necessarily static components but are best understood as different *roles*, for which the same components may be re-used in multiple contexts.

Introduction

In Autonomic Computing (Ganek & Corbi 2003), "self-protection" is one of the "self*" properties which enables an autonomous system to recover from faults and intrusions without external intervention. Such a system requires a meta-level to recognise and correct problems in its own operation without external intervention. The same applies to an autonomous robot which must survive in a hostile environment.

The simplest design for an autonomous self-protecting system is hierarchical with a central meta-level ensuring that the system operates according to requirements. However, this design is vulnerable because the meta-level cannot monitor its own operation independently.

In earlier work (Kennedy & Sloman 2002; 2003; Kennedy 2003) we developed some implementations in which distributed meta-levels monitor and repair each other to overcome the vulnerabilities of a hierarchical system. A distributed self-model is acquired by a process of bootstrapping where the different components learn by mutually observing each others' internal processes. However, in these implementations the higher level cognitive capabilities were not developed in detail, since the focus was on survival and self-repair. In this paper, we extend the previous work by showing how distributed meta-levels can be integrated into a full cognitive architecture required for human-like cognition. In particular, we argue that distributed meta-level architectures are also capable of *self-explanation* as emphasised in (Cox 2007).

The structure of the paper is as follows: we first define a full cognitive agent with centralised meta-management. We then extend this basic cognitive agent so that it has multiple meta-levels which are distributed and mutual. In the third part, we will show how the distributed meta-levels relate to the concept of self-explanation, and argue that this is possible using some examples. Finally, we consider the need for global coordination which is required for self-explanation and action.

Related Work

The emergence of collective self-awareness in distributed systems such as immune systems and ant colonies is discussed in (Mitchell 2005). Artificial immune systems which distinguish between "self" and "nonself" are a major inspiration for our work. If our distributed meta-level architecture were applied to a large number of components, it may behave globally like an immune system on a high level. However, such artificial immune systems such as (Hofmeyr & Forrest 2000) currently do not provide a clearly defined meta-level and object-level relationship which is necessary to understand and predict the interactions between the components of the system on a microscopic level. Such predictability and assurance of correct operation is usually required in autonomic computing. Furthermore, these systems in their current form are limited in their transparency and self-explanation capability.

Our architecture is similar to a multi-agent meta-level control system, such as in (Raja & Lesser 2007). The main difference is that our system is non-hierarchical, since all meta-levels also play the role of object-levels. Additionally, we are primarily interested in a distributed control system for a single agent. The architecture could be applied to multiple agents, but they need the ability to access (and modify) each other's mental states and internal processes. The different meta-levels in our architecture can play a similar role to the "reflective critics" in the architecture of (Singh 2005), which are applied to the agent's mental states.

A Cognitive Architecture with Meta-management

As a starting point for distributed meta-management, we define a cognitive agent with a single meta-level, based on the main features of the H-Cogaff architecture of (Sloman 2001). H-Cogaff has three layers: (a) a *reactive* layer, which responds rapidly to events in the environment but with very little thought, (b) a *deliberative* layer, which uses its knowledge of the environment to reason about hypothetical scenarios and to plan ahead, and (c) a *meta-management* layer (Beaudoin 1994; Hansen & Zilberstein 2001; Cardon *et al.* 2001) which critically evaluates the deliberative layer and interrupts it if necessary (e.g. if it is not making progress).

Our simplified version of the H-Cogaff architecture is shown in Figure 1. This shows an agent as a two-layer structure containing an object-level O_1 and a meta-level M_1 respectively.

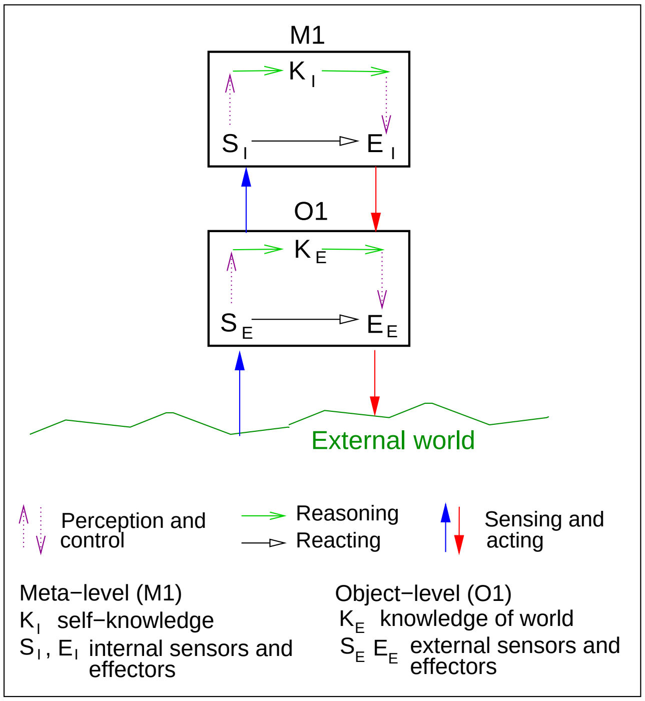

Figure 1: A cognitive agent with meta-management

On the object-level, K_E represents knowledge of the external world to allow deliberative reasoning. Some of this can be a predictive model (e.g. involving rules about expected behaviour of objects). O_1 also includes a reactive layer, which involves immediate reactions to the external world without the use of a high level representation or hypothetical reasoning (the arrow directly from sensors to effectors). The dotted vertical arrows within boxes are translations between levels of abstraction. "Perception" is a translation from sensor readings to high-level knowledge. Similarly, "control" is a translation from the selected options on the knowledge level into motor sequences.

Meta-level The meta-level in Figure 1 monitors and critically evaluates the information states and processes that the object-level is *relying on* (e.g. its knowledge and algorithms).

For example, the deliberative layer in O_1 could enable the agent to predict the effect of its own actions using its knowledge K_E (e.g. "if I move the red block onto the blue block, the top of the blue block will be covered"). In contrast, the meta-level does not rely on K_E but can question it. For example, it might predict the agent's ability to understand the relationship between the blocks (e.g. "my knowledge of 'blocks' and 'surfaces' is limited and I might get confused"). For humans, the equivalent is reasoning about mental states.

In addition to mental states, the meta-level reasons about informational entities which *refer* to entities in the external world (indicated by single quotes for 'blocks' and 'surfaces' above). This means that the meta-level also needs to know the state of the world in which informational processes take place so that they can be evaluated according to the progress made towards a goal.

Internal sensors and effectors (S_I and E_I) are used on the meta-level to detect and modify object-level states and processes. Meta-level monitoring requires some form of log or trace of activity (Cox 2007). Internal effectors can modify any object-level components or their configurations (e.g. priorities for sensing) or initiate new learning processes.

The agent's knowledge about its information processing is represented in K_I, which contains a map of object-level components, their current states and a model of their normal behaviour which is learned through a process a self-familiarisation. A "component" might be a software component or executing process or it can be an abstract concept (e.g. "focus of attention") used within an ontology of mental states.

Comparison with Cox and Raja's "Duality in Reasoning and Acting". (Cox & Raja 2007) have defined a meta-level architecture representing "Duality in reasoning and acting" which is reproduced in Figure 2. Our architecture can be regarded as a specific instance of this general class of systems. The object-level in Figure 1 (containing both reactive and deliberative layers) corresponds to the object-level

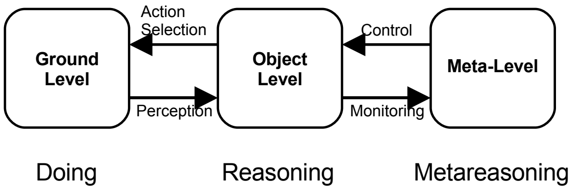

Figure 2: Duality in reasoning and acting (Cox and Raja, 2007)

in Figure 2 and the environment in Figure 1 corresponds to the ground level in Figure 2.

Anomaly Detection

An "anomaly" is an unexpected occurrence that may indicate a problem. In Figure 1, K_I contains "self-knowledge" acquired through earlier self-observation and allows the meta-level to predict the pattern of reasoning of the object-level, given a known environmental event. Effectively the agent anticipates its own mental state.

One kind of anomaly is when the reasoning process does not follow a desirable pattern, despite events in the world appearing to be normal. For example, the reasoning may be focused on irrelevant details. The reason for this problem might be distraction due to other pressures (e.g. competing goals).

Another kind of anomaly is when the model of the world contained in K_E predicts a state that does not correspond to the reality. The recognition of the "strange" nature of the new situation is an introspective (hence meta-level) function because it involves questioning whether the agent's current knowledge is sufficient for the situation. The diagnosis process should determine whether any new learning is required or whether the anomaly is due to other factors such as a failure of a sensor interpretation component.

Distributed Meta-Management

In the self-protection context, a hostile environment can attack *any* part of the system including the failure detection and recovery components of the meta-level. For example, the meta-level failure detection code may be illegally modified, or some of its reasoning components may be prevented from executing.

A more complex kind of failure is a lack of knowledge or inaccurate knowledge in the model used by the meta-level to detect problems in the object-level. The meta-level relies on its knowledge (K_I) of the object-level in the same way as the object-level relies on its knowledge (K_E) of the world. Questioning its own model requires an additional meta-level (according to the definition we are using).

Instead of adding an infinite regress of meta-levels (homunculus problem), additional meta-levels can be distributed so that they can mutually monitor and evaluate each other. The aim is to critically question and independently evaluate each reasoning process on all levels. Although in practice, some gaps in coverage will exist, all components that are *essential for the survival of the agent* must be protected and their reasoning subjected to critical evaluation.

(Cox & Raja 2007) have defined a form of distributed meta-reasoning as a multi-agent architecture in which the meta-levels of the respective agents exchange information in order to coordinate their actions. Cox and Raja's diagram is reproduced in Figure 3. Our approach is different from this multi-agent architecture, because we are aiming for distributed meta-level control (of one agent), where the inter-meta-level relationship is mutual; in other words, both meta-levels are also each other's "object-level". All meta-levels are monitored and protected by at least one remaining meta-level to provide full self-protective coverage. One particular configuration of this is shown in Figure 4. In this case, two meta-levels monitor each other, and both monitor the same object-level. The relation between M_2 and M_1 has the same nature as that of M_1 and O_1. The main difference is that the interaction between M_2 and M_1 is two-way (since M_1 also monitors M_2). The actions, however, need to be coordinated so that meta-levels do not interfere negatively with each other. This is indicated by dashed arrows pointing away from the boxes, meaning that the actions are not always executed (or not immediately).

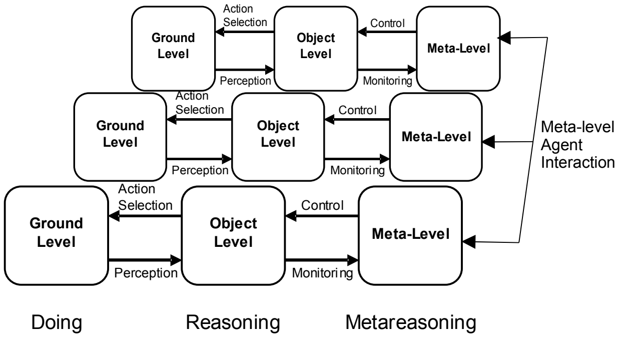

Figure 3: Distributed meta-levels (Cox and Raja, 2007)

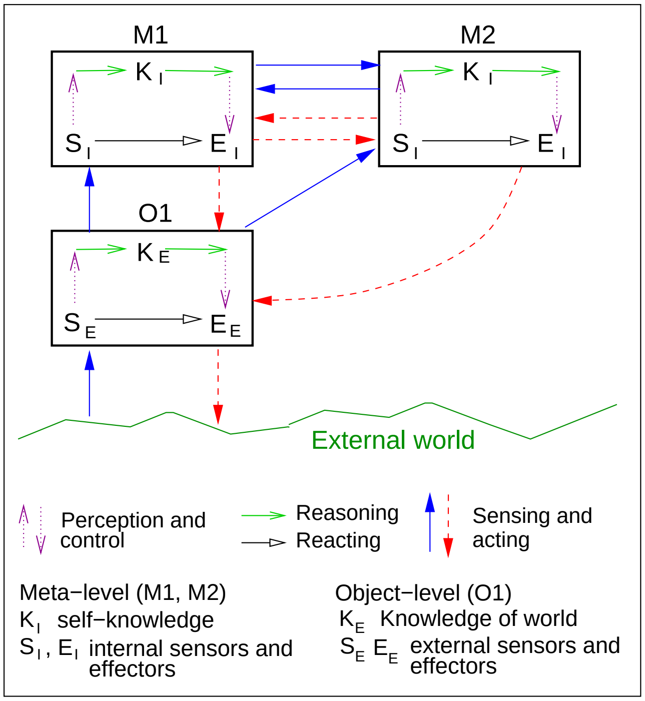

Figure 4: Distributed meta-management

A different configuration is shown in Figure 5, where the two object levels are alternative ways of representing and reasoning about the world. The simplest architecture is where one of them is the "primary" with control over actions and the other is the "backup" which is ready to take control if the first one fails (classic fault-tolerance architecture). In the diagram, the sensors and effectors are separate for each object-level, but they may be shared.

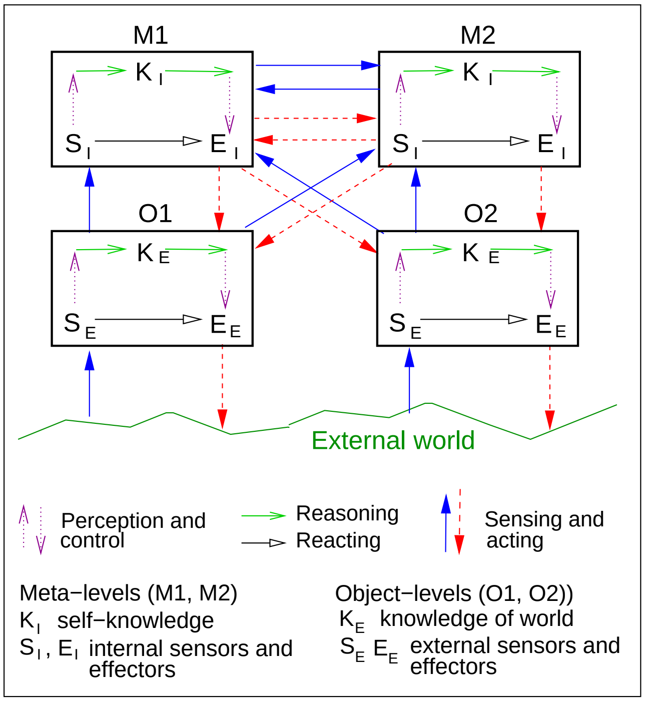

Figure 5: Distributed meta-management with different object-levels

Monitoring is concurrent The meta-levels need to be active concurrently. To detect problems in M_1's operation, M_2 should be able to dynamically adjust its monitoring of M_1's reasoning in response to any problems that it finds. The same is true for M_1's monitoring of M_2. The alternative is a situation where the control flow changes sequentially from object-level reasoning (O_1) to M_1 (which reasons about O_1 and then reasons about M_2) to M_2 (which reasons about O_1 and then reasons about M_1) and back. This is inefficient and not very adaptive because it cannot adjust the monitoring dynamically in response to observed changes. Most importantly, it would also be vulnerable: the failure in O_1 may be a failure that shuts down all meta-levels above it (or it simply refuses to hand over control to the next one). It is important that each level operates concurrently and its execution does not depend on other levels.

Mutual and concurrent monitoring is different from mutual and concurrent *action*. For self-protection, a meta-level must intervene to modify the operation of other components (in a meta-level or object-level). In most application domains, however, we expect the actions to be coordinated and sequential. This is why the actions in the diagrams are shown as dashed arrows, because they are not expected to be active without coordination.

Closed Meta-level Networks

Each meta-level uses its internal sensors and predictive model to monitor the meta-level of the other. The aim is to ensure that all meta-levels are themselves monitored, and that the monitoring is sufficiently accurate for the situations the agent must survive in. We expect that a "closed" architecture of this kind will be less vulnerable to the failure of a meta-level, and that important gaps in knowledge that are causing problems can be detected and overcome.

Meta-level roles Although it is convenient to draw a diagram with separate boxes for object-levels and meta-levels, they are not necessarily separate components. They are best thought of as different *roles*. The same component can be playing the role of meta-level and object-level simultaneously (because of the 2-way relationship).

Concurrent monitoring means that a meta-level role is not triggered by any event. However, increased meta-level processing may be triggered if a meta-level detects an anomaly in an object-level (or other meta-level). For example, M_2 may vary the amount of data that it collects on M_1 so that the trace is more detailed.

Minimising additional complexity In Figure 4 the "horizontal" inter-meta-level relation has the same form as the vertical relation between meta-level and object-level (the same kind of arrows are used). The goal is to minimise the additional features required for distributed meta-level processing by using the same mechanisms that are already used in the meta-object-level relation. These same mechanisms (e.g. rules or sections of code) are concurrently playing different roles. They are "talking about" different things, but *using the same concepts and methods*. This is similar to the software engineering practice of component re-use in multiple contexts (see for example, (Northrop 2002)).

It is also necessary to limit meta-level complexity in order for the system to become familiar with the correct operation of its meta-levels, which is required for detecting their failure. We have shown that this is possible in a proof-of-concept implementation in (Kennedy 2003). In that implementation, the architecture was "multi-agent" and corresponds to a very simplified version of Figure 5, except that the object-levels are identical copies and the meta-levels did not directly access or intervene in another agent's object-level. However, the basic principles are the same as for Figure 4 since the meta-levels could access and modify each other's processing directly. In the next section, we summarise briefly the scenario and method of learning.

An Example Scenario

In the previous work, we designed a simple virtual world called "Treasure", implemented in SimAgent(Sloman & Poli 1995). An autonomous vehicle collects treasure, while avoiding ditches and ensuring that its energy supply is regularly recharged. The collected treasure gradually loses value and must be maintained above a minimal level or the system "dies".

In addition to simulated threats from the virtual world, components of the agent controlling the vehicle can fail or be subverted by an enemy. For example, the perception of the agent can fail so that it interprets the environment wrongly (e.g. it does not see a nearby ditch). Its failure detection and recovery components can also fail or be subverted. Such faults were artificially injected to test the system. We successfully implemented a form of "reactive repair", where a faulty component would be detected and "repaired" immediately by replacing it with a trusted backup component.

Traces available to a meta-level In SimAgent, each agent is implemented as a set of "condition-action" rules, which is divided up into modules (rulesets). Each module is concerned with sensing, perception, reasoning, acting or other functions (such as meta-reasoning).

In the Treasure implementation, the meta-level's model of expected internal processing (corresponding to K_I) is a set of patterns of rule-firing traces associated with different activities. This was learned during a self-observation phase in a protected environment. During normal operation in a non-protected environment allowing intrusions, this model can be used as a guideline for the correct behaviour of object-level components. The agent can access a trace of recently fired rules (corresponding to internal sensors). If the actual rule firing pattern does not fit the model-predicted pattern, and is accompanied by an unexpected worsening of the external world (which we called "hostility" detection (Kennedy & Sloman 2003)), the meta-level decides that a failure in the affected set of rules has occurred. This is comparable to the episodic memory required by (Cox 2007), except that the Treasure implementation does not involve a detailed semantic or linguistic interpretation of these traces, since it is a "shallow" cognitive system.

Mutual bootstrapping of models The model of expected behaviour was "bootstrapped" by discovering classes of activity in rule-firing traces (Kennedy & Sloman 2002). For an architecture with two meta-levels, we divided the training into three phases:
Phase 1: Both meta-levels learn about object-level processing:

- Each M_i (where i is 1 or 2) learns about the normal operation of the object level in a protected environment and builds its initial model K_I.
- The object-level interacts with the environment in situations it has learned previously.

Phase 2: Both meta-levels take turns observing each other detect and respond to attacks against an object-level in an environment permitting object-level attacks only. For a meta-level M_j learning about M_i, this looks as follows:

- M_j detects failure in the object-level and takes corrective action using K_I.
- M_i observes the trace of M_j's reasoning and activity during its error detection and repair of the object-level and uses these traces to build its model of M_j (part of its K_I).

Phase 3: Operational phase with no restriction on hostile attacks, where a meta-level M_i is repaired by M_j:

- M_j detects failure in M_i using its model of correct operation of M_i acquired during Phase 2;

In the above, we are assuming that the reasoning processes and actions of meta-level M_j when detecting a failure in M_i will be approximately the same as M_j's processing when detecting a failure in the object-level (which M_i has learned). Component re-use should make this assumption reasonable. This is also similar to the kind of assumption made about training scenarios in general; they should be a good enough approximation of reality to be useful.

Towards Self-Explanation in a Distributed Meta-management System

Explanation is often necessary for self-protection. In many cases, a simple "reactive repair" of a problem (as implemented in the Treasure scenario) will not be sufficient. It will be necessary to explain the cause of the failure and plan corrective actions accordingly. In addition, self-explanation is an important requirement for general meta-cognition (Cox 2007).

It is easy to see how a self-explanation capability can enhance an agent with a single meta-level. However, there are problems with self-explanation in a mutual meta-management system. First, a translation is required from multiple *objectively* defined interactions of distributed software components into a single *subjective* narrative about the agent's own mental states. Secondly, the global coordination needed to provide coherent actions and explanations needs to be reconciled with a decentralised architecture in which no single component is in control.

We will consider first some examples of meta-levels giving competing explanations and then go on to show how the potential problems may be overcome. While discussing each example in the context of an objectively defined architecture, we will also consider what a natural language explanation might look like from a human meta-cognitive point of view. In particular, its capability to chain together the different states of the whole system into a coherent and global narrative is important.

Examples of Competing Meta-level Explanations

To show that it makes sense for meta-levels to reason about each other and to explain different kinds of failure, we first consider some examples of a meta-level reasoning about an object-level and then apply similar examples to one meta-level reasoning about another. We will refer to Figure 4 throughout, although the examples should also apply to other configurations. Finally, we consider what problems arise when mutual meta-levels make conflicting statements about an object-level or about each other.

Example 1: Explanation involving object-level only The following are example explanations by M_1 about a failure of the object-level O_1, once the problem has been diagnosed by M_1:

- Example 1.1: Simple failure detection: "O_1 failed because step S was never executed" For example, component S may be an algorithm that O_1 was relying on to interpret sensor readings.

- Example 1.2: Reasoning about errors and intrusions: "O_1 delivered a wrong result because of an error in the reasoning at step S".
- Example 1.3: Detection of lack of knowledge (or incorrect knowledge): "O_1's knowledge of the world is insufficient: its model predictions do not agree with the reality".

Comparison with human meta-cognition In the case of Example 1.1, a human-like robot with meta-cognition might say: "I forgot to do S". In Example 1.2, it might say: "I got confused during step S because I was distracted by a different problem", which is similar to the hostile intrusion problem. Another kind of explanation might be: "I made a mistake during step S because I have a tendency to make this kind of mistake in these circumstances". This is similar to a design error, which can be corrected by further learning or self-modification. The robot's self-knowledge (what the "I" stands for) is in this case focused on M_1's knowledge of O_1 only.

A cognitive robot equivalent for Example 1.3 is easier using a concrete scenario. If a robot intends to lift a tea cup from a table, its model of the normal properties of tea cups (part of K_E) predicts that it will be lifted easily (because it is fairly light). If, however, the robot is in an unusual situation where the cup is stuck to the table, it finds that the outcome of the lifting operation contradicts the model predictions. It may explain this by recognising its own *lack of knowledge* about the new situation, and that it needs to learn more. For example, it may decide to explore by attempting to find out more details about the table surface or the cup surface. Mini-scenarios of this kind have already been discussed in (Minsky, Singh, & Sloman 2004).

Example 2: Distributed meta-levels M_2 reasoning about M_1 is analogous to M_1 reasoning about O_1. However, it is unlikely that each meta-level will only be monitoring the one below, since it is necessary to understand the *context* in which errors occur. Therefore, we assume sensors and effectors of *both* meta-levels can access the same object-level (shown in Figure 4), although this is only one possible configuration. The following are three examples of M_2 explaining a failure of M_1:

- Example 2.1: "M_1 failed to detect the error in O_1's reasoning at step S (because of an error in M_1's execution of step T in its own processing)". In this case M_1 is "fail-silent" (fails to operate). Note that M_2 monitors two different levels and may be able to explain failures on both levels.
- Example 2.2: "M_1 wrongly detected an error in O_1 at step S; the problem is in step U". This introduces an additional problem of *disagreement* between two meta-levels, since M_1 is also reporting a problem. Furthermore, because of the mutual relationship, M_1 may also claim to have detected an error in M_2. We return to this problem below.
- Example 2.3: "M_1's knowledge of O_1 needs to be corrected or extended". The same problem of disagreement may also occur here.

Distributed meta-levels and human meta-cognition Distributed meta-levels can also have an equivalent in human-like meta-cognition. Example 2.1 might be expressed as: "How did I fail to notice my mistake at step S? I am not being cautious enough". This is a criticism of one's own meta-management, due to its failure to do anything at all ("fail-silence").

Example 2.2 is more complex, but might have the following form: "I suspect I made a mistake during step S as it does not seem to have gone correctly (this is M_1's hypothesis because the trace of step S appears unusual), but I have to keep open the possibility that it is something different - maybe step S was correct but actually step U is wrong and I'm just thinking incorrectly that S was wrong (M_2's critical questioning of M_1's interpretation)". Here, the meaning of the term "I" changes from M_1 to O_1 and then to M_2 in the course of the sentence. The relationship between the different levels and the linguistic self-explanation is made clearer in Table 1. In this example, a disagreement between two meta-levels can result in indecision. Example 2.3 is similar, and may have the following form "I suspect I'm not understanding this because I don't know very much about concept C, but it is also possible that my knowledge about C is correct and I'm just thinking wrongly that I'm not understanding it." M_2's hypothesis is that M_1's understanding of what O_1 needs to know is wrong. In this case, it may be possible to explore both competing hypotheses.

Part of sentence	Meaning of "I"
"I suspect that .."	M_1
"I made a mistake ..	O_1
"but I have to keep open .. "	M_2

Table 1: Changing meaning of "I" during self-explanation

Ontologies of Mental States

In the examples above (in particular, 2.2 and 2.3), M_2 needs a different kind of model of O_1 than the one used by M_1. Otherwise, it has no grounds for questioning M_1's knowledge (since they would both use the same knowledge and methods). The two meta-levels may share the same ontology of mental states but use different methods or algorithms for monitoring and reasoning about them, which can lead to competing beliefs and hypotheses expressed using the same concepts.

Another possibility is that both meta-levels use different ontologies because they are focusing on different aspects of O_1's processing. For example, instead of focusing on the pattern of reasoning (such as a rule firing trace) M_2 may inspect a series of snapshots of the short-term memory contents associated with both O_1 and M_1 and how they changed. Thus M_1 might cause a robot to ask: "why did I choose this action?" (which might be explained in terms of rules which matched the current state), while M_2 causes it to ask "how did my beliefs and goals change over time?" (which is mainly about the focus of attention). Current research on ontologies of mental states includes (e.g. (Ferrario & Oltramari 2004).)

The different types of model can be learned using the mutual bootstrapping process outlined earlier, except that the detailed methods and representations will be different. Whether the ontology is centred on rule-firing or memory content, the relevant traces can be grouped together into clusters representing different classes or "modes" in which the agent finds itself while executing various tasks (including meta-level tasks).

Coordination of Explanation and Action

Each meta-level can explain a failure from its own point of view, and such a local explanation can be sufficient for it to plan a "repair" action. However, many situations require a sequential coordinated action where the different components must cooperate (e.g. in robotics). Furthermore, disagreement can occur between meta-levels about the diagnosis and the correct action plan (as in Example 2.2). In such cases, actions may interfere with each other negatively, thus blocking any progress. The need for coordinated action is the same as that for a single coordinated explanation.

Avoiding oscillations and deadlock If different meta-levels detect different kinds of failure, negative interference may be prevented by ensuring that the first meta-level to make a decision on how to act will inhibit all others (e.g. by broadcasting that it has made a decision). Since the remaining meta-levels cannot evaluate the "winning" meta-level's action until it has made some progress, oscillations should be prevented. Longer term oscillation may of-course still occur, but for complex real-world problems, human cognition is also subject to indecisiveness and "false starts".

Deadlock should also be prevented, since a meta-level can interrupt another one that is exclusively accessing a resource (e.g. if there is lack of progress). This means that one of the four conditions for deadlock ("no preemption") does not hold.

In a self-protective scenario in a hostile environment, a meta-level may contain hostile code and cause rapid damage (not just fail to correct a problem). Avoiding oscillation is more challenging in this case. A classic fault-tolerance approach is to use a majority voting system to determine whether to allow a component to take action. Distributed agreement is a non-trivial problem, but significant progress has been made in this field. For an overview, see (Verissimo & Rodrigues 2001).

Global self-explanation helps coordination For coordination of action, a single global explanation of the state of the whole system, which takes into account different meta-level viewpoints is useful. This is why it is helpful to consider human meta-cognitive explanations, since they seem to do what is required.

Human-like self-explanation may be provided by a meta-level that collects information from all other meta-levels and object-levels and then links their belief states together to form a coherent narrative. This requires a global and summarised overview of the whole system, such as in "global workspace" theory (Baars 1988) although the content of the global workspace may be changing during the course of the explanation. The implementation in (Singh 2005) includes a "meta-managerial" critic, which has a similar global function.

More than one meta-level may participate in constructing a global narrative. In the same way as for action coordination, the first meta-level to construct an atomic part of an explanation (e.g. a sentence) can broadcast its readiness to act and inhibit all others. Subsequent sentences can be constructed by others.

To satisfy the requirement of closed meta-levels (where all meta-levels are monitored), a meta-level constructing an explanation must itself be subject to critical evaluation in the same way as other meta-levels. This would be analogous to human meta-cognition, where it is possible to doubt one's own self-explanation and interrupt it. Oscillations are avoided because interruptions only happen after some delay. As with all boxes in the diagrams, such a meta-level does not have to be a static component in the architecture, but instead an emergent stable coalition of participating components as in some neuroscience models (e.g. (Koch 2004)).

Cost of Distributed Meta-Management

From the above, it is clear that meta-management has potential disadvantages due to its added complexity. This cost can be measured in several ways:

- Processing time: for all meta-level functions (including monitoring, reasoning, coordination, communication and action), how much additional computing time is required when compared to object-level processing on its own?
- Problem-solving cost: how much does the meta-level processing interfere with the problem being solved by the object-level? In particular, for meta-level interventions and control, how much do incorrect interventions, such as learning something irrelevant have a detrimental effect on the object-level task?

For distributed meta-management, the additional costs may be offset by opportunities for adaptaton and self-optimisation that would not be possible in a centralised meta-level system. We discuss these challenges below.

Complexity of self-familiarisation Coordination mechanisms add complexity to the meta-levels, which makes them in turn more difficult to be monitored. In (Kennedy 2003) we demonstrated that the mutual bootstrapping method summarised earlier can also be applied to three meta-levels where each one monitors the remaining two. During training, the meta-levels must also become familiar with the additional coordination required for majority voting because this is part of their correct behaviour. This makes the training phase more complex because each meta-level has to activate the reasoning patterns and actions that *would be active* in an inter-meta-level coordination system. Although this worked successfully in a restricted artificial environment, it remains a significant challenge for large real-world sytems.

The self-familiarisation phase would need to take place in parallel instead of sequentially as in our test scenario.

Connectivity Figure 5 may be generalised to include n object levels and m meta-levels, each able to monitor and change the other. Since full connectvity would be inefficient, only those connections that are the most useful should be preserved. Unproductive connections between meta-levels and object-levels may be disabled if a sufficient number of meta-levels agree that a connection has been unproductive or detrimental to the object-level task. No additional complexity is required here, other than the mechanisms already in place for agreement about meta-level interventions. The system may eventually stabilise into a configuration that counteracts the overheads of a single meta-level system, where disruptive operations are not corrected.

Conclusions and Future Work

In the context of autonomic computing or robotics, the capability of the system to protect itself against faults and intrusions requires a non-hierarchical distributed architecture. However, self-explanation is also important, not only for the system itself but also to enable humans interacting with it to understand the reasons for its actions. Reconciling these two requirements is possible, but requires an integrated, cross-disciplinary approach to cognitive systems. In addition to AI methods, research in distributed fault-tolerance, software engineering and cognitive neuroscience can make a valuable contribution.

References

Baars, B. J. 1988. *A Cognitive Theory of Consciousness*. New York: Cambridge University Press.

Beaudoin, L. P. 1994. *Goal Processing in Autonomous Agents*. Ph.D. Dissertation, University of Birmingham.

Cardon, S.; Mouaddib, A.; Zilberstein, S.; and Washington, R. 2001. Adaptive Control of Acyclic Progressive Processing Task Structures. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, 701–706.

Cox, M. T., and Raja, A. 2007. Metareasoning: A Manifesto. Technical Report BBN TM 2028, BBN Technologies.

Cox, M. T. 2007. Metareasoning, Monitoring, and Self-Explanation. In *Proceedings of the First International Workshop on Metareasoning in Agent-based Systems at AAMAS-07*, 46–60.

Ferrario, R., and Oltramari, A. 2004. Towards a Computational Ontology of Mind. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS 2004)*, 287–297. IOS Press Amsterdam.

Ganek, A. G., and Corbi, T. A. 2003. The Dawning of the Autonomic Computing Era. *IBM Systems Journal* 42(1):5–18.

Hansen, E., and Zilberstein, S. 2001. Monitoring and Control of Anytime Algorithms: A Dynamic Programming Approach. *Artificial Intelligence* 126(1-2):139–157.

Hofmeyr, S. A., and Forrest, S. 2000. Architecture for an Artificial Immune System. *Evolutionary Computation* 8(4):443–473.

Kennedy, C. M., and Sloman, A. 2002. Acquiring a Self-Model to Enable Autonomous Recovery from Faults and Intrusions. *Journal of Intelligent Systems* 12(1):1–40.

Kennedy, C. M., and Sloman, A. 2003. Autonomous Recovery from Hostile Code Insertion using Distributed Reflection. *Journal of Cognitive Systems Research* 4(2):89–117.

Kennedy, C. M. 2003. *Distributed Reflective Architectures for Anomaly Detection and Autonomous Recovery*. Ph.D. Dissertation, University Of Birmingham, Birmingham, UK.

Koch, C. 2004. *The Quest for Consciousness: A Neurobiological Approach*. Englewood, Colorado: Roberts and Company Publishers.

Minsky, M.; Singh, P.; and Sloman, A. 2004. The St. Thomas Common Sense Symposium: Designing Architectures for Human-Level Intelligence. *AI Magazine* 25(2):113–124.

Mitchell, M. 2005. Self-Awareness and Control in Decentralized Systems. In *Working Papers of the AAAI 2005 Spring Symposium on Metacognition in Computation*, 80–85.

Northrop, L. M. 2002. SEI's Software Product Line Tenets. *IEEE Software* 19(4):32–40.

Raja, A., and Lesser, V. 2007. A Framework for Meta-level Control in Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems* 15(2):147–196.

Singh, P. 2005. *EM-ONE: An Architecture for Reflective Commonsense Thinking*. Ph.D. Dissertation, Artificial Intelligence Lab, MIT.

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

Sloman, A. 2001. Varieties of Affect and the CogAff Architecture Schema. In *Symposium on Emotion, Cognition, and Affective Computing at the AISB'01 Convention*, 39–48.

Verissimo, P., and Rodrigues, L. 2001. *Distributed Systems for System Architects*. Dordrecht, The Netherlands: Kluwer Academic Publishers.