

# 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.

## 1 Introduction

In Autonomic Computing [6], “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 [9, 10, 8] 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 [3].

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.

## 2 Related Work

The emergence of collective self-awareness in distributed systems such as immune systems and ant colonies is discussed in [13]. Artificial immune systems which distinguish between “self” and “non-self” 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 [7]) 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 [15]. 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 [16], which are applied to the agent’s mental states.

Some research outside of AI is also relevant. Many of the challenges arising from the need to coordinate meta-level actions and enable their participation in global self-explanations have been addressed in distributed fault-tolerance. An overview is in [19]. Furthermore, the need to minimise the additional complexity as a result of distribution and coordination of meta-levels is related to the need to reduce complexity in software engineering, which has been addressed by the *re-use* of the same components and patterns in different contexts (see e.g. [14]).

## 3 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 [18]. 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 [2], which critically evaluates the deliberative layer and interrupts it if necessary (e.g. if it is not making progress). “Meta-management” is a particular kind of meta-level that involves the ability of a cognitive system to detect problems in its internal processing and modify them. It can include lower level autonomic control operations in addition to higher level reasoning about mental states.

### 3.1 A simplified version of H-Cogaff

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 and a meta-level respectively. The object-level  $O_1$  carries out some primary task in the external world, such as exploring hazardous terrain and collecting data.

On the object-level,  $K_E$  represents knowledge of the external world which includes a model of its normal behaviour. Some of this can be a predictive model (e.g. involving rules about expected behaviour) which can allow internal “simulation” by executing the rules in hypothetical scenarios (“what-if” reasoning), which is an important part of deliberation.  $O_1$  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 (which may be a plan) into motor sequences (control of effectors).

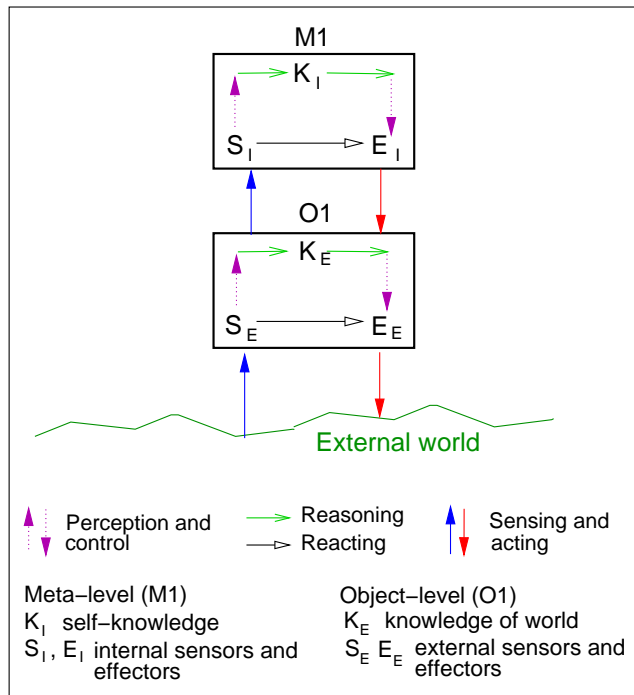


Figure 1: A cognitive agent with meta-management

### 3.1.1 Meta-level

The meta-level in Figure 1 monitors the reasoning process and actions of  $O_1$  and evaluates their “quality” according to the progress made towards a goal in the external world.

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 [3]. 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 itself is represented in  $K_I$ , which contains a map of object-level components, their current states and a model of their normal behaviour. It is similar to  $K_E$ , except that  $K_I$  is a representation of *internal processes* involved in reasoning about the external world and reacting to it, not the external world itself. 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.

### 3.1.2 Comparison with Cox and Raja’s “Duality in Reasoning and Acting”.

Cox and Raja[4] 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

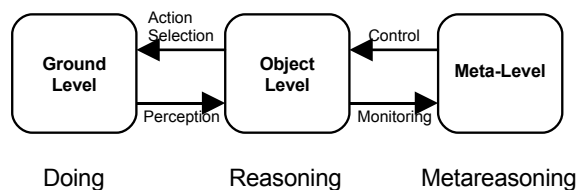


Figure 2: Duality in reasoning and acting (Cox and Raja, 2007)

general class of systems. The object-level in Figure 1 (containing both reactive and deliberative layers) corresponds to the object-level in Figure 2 and the environment in Figure 1 corresponds to the ground level in Figure 2.

An additional feature in Figure 1 is that the meta-level can be characterised as also having (internal) sensors and effectors, except that they are processing *internal* events. Some of these internal events may relate to the high-level reasoning pattern (for which an ontology of mental states may be used) while others may be concerned with low-level “autonomic” processes, to which the system reacts to and adjusts quickly (as is often required in autonomic control systems). In each box, the reactive and deliberative layer may be closely coupled.

### 3.2 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 and actions of the object-level, given a known environmental event. Effectively the agent anticipates its own mental state.

One kind of anomaly that can be detected by the meta-level is when the reasoning process does not follow a desirable pattern, despite events in the world appearing to be normal. In other words, it appears that the model of the world is in agreement with sensor readings, but the reasoning about them is not making any progress or it seems to be focused on details that are not central to the task. The reason for this problem might be distraction due to other pressures (e.g. competing goals) or confusion due to a failure of a sensor interpretation component.

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 (as detected by external sensors). The recognition of the “strange” or “unknown” 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 or whether it needs to extend its model. For example, it should determine whether the cause of the anomaly is a fault in the object-level as above (e.g. perception is interpreting the environment wrongly and getting confused) or whether in reality a new kind of situation exists. The diagnosis process should determine whether any new learning is required, and what to learn.

Note that in both cases, the meta-level always trusts its own model of the object-level, but it may doubt the object-level’s model of the world.

## 4 Distributed Meta-management

We now show how to extend the basic cognitive architecture to include 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. Therefore, the meta-level knowledge may also have gaps.

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 and Raja[4] 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

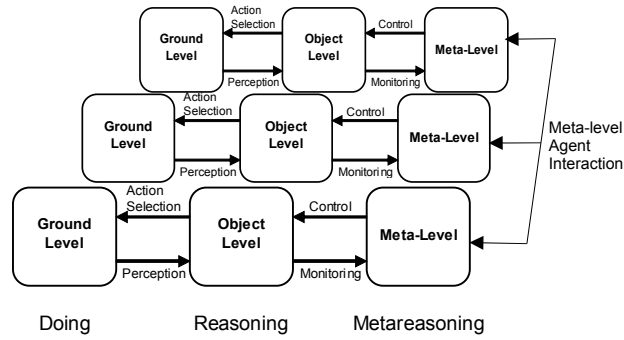


Figure 3: Distributed meta-levels (Cox and Raja, 2007)

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,

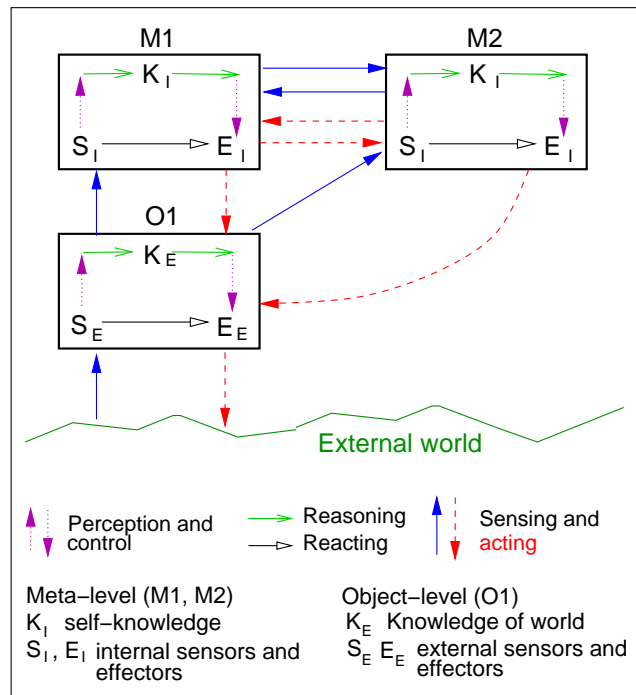


Figure 4: Distributed meta-management

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  $M_1$  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).

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 may be generalised to include  $n$

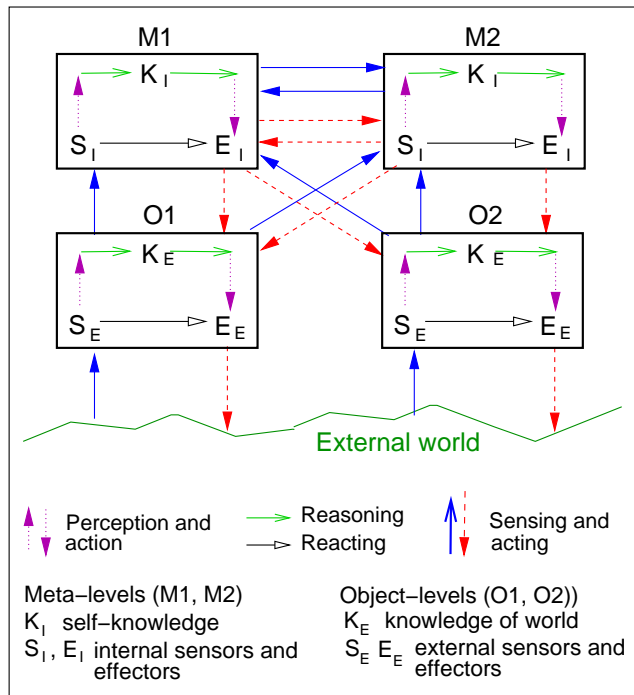


Figure 5: Distributed meta-management with different object-levels

object levels and  $m$  meta-levels, each able in principle to monitor and change the other. However, in a realistic system, full connectivity would not be efficient. Instead, those connections that are found to be the most useful are preserved, which is analogous to the way that natural brains evolve and adapt.

#### 4.1 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, although there may be some exceptions where meta-levels may be permitted to influence each other simultaneously. The simple primary/backup configuration in Figure 5 is one solution. We will return to this problem in the next section.

#### 4.2 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.

#### 4.2.1 Levels are not separate components

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). Similarly, the same sensors and effectors may be shared by different “boxes”. There is, however, an important difference between *internal* and *external* knowledge (as well as the respective sensor and effectors).

#### 4.2.2 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, [14]).

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 [8]. In that implementation, the architecture was “multi-agent” and corresponded 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 background and method of learning by self-observation which we used in [8] in order to develop the internal models used by each meta-level.

### 4.3 An example scenario

In the previous work [8], which was focused on reactive self-protection inspired by the immune system, we designed a simple virtual world called “Treasure”, implemented in SimAgent[17]. 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, or its energy sensor is faulty). 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. (The need for more deliberative forms of diagnosis and repair will require the ability to *explain* what went wrong which we will discuss later.)

#### 4.3.1 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). During each simulation cycle, the sensors of each agent are updated, their rulesets are executed and the effect of agent actions on the environment is updated.

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 [10]), the meta-level decides that a failure in the affected set of rules has occurred. This is comparable to the episodic memory required by [3], except that the Treasure implementation does not involve a detailed semantic or linguistic interpretation of these traces, since it is a “shallow” cognitive system. However, these capabilities may be added as part of architectural deepening.

### 4.3.2 Mutual bootstrapping of models

The model of expected behaviour was “bootstrapped” by discovering classes of activity in rule-firing traces [9]. 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.

## 5 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 [3].

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 de-centralised 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.

## 5.1 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.

### 5.1.1 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 (or a component that S relies on failed)” For example, component S may be an algorithm that  $O_1$  was relying on to interpret sensor readings). S can be a small step in a procedure, or it can itself be a complex procedure requiring its own specialist knowledge.
- Example 1.2: Reasoning about more general 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”. Note that this also may be a failure in the sensors or in the process of making predictions (e.g running predictive models), and not in the actual knowledge.

### 5.1.2 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 (e.g. sudden sensory overload)”, 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 [12].

### 5.1.3 Example 2: Distributed meta-levels

$M_2$  reasoning about  $M_1$  is analogous to  $M_1$  reasoning about  $O_1$ . For example, if the object-level  $O_1$  relies on components to interpret external sensor readings, the first meta-level  $M_1$  relies on similar components to interpret traces of  $O_1$ 's internal processing. Such a component might be wrongly interpreting  $O_1$ 's reasoning traces and making an incorrect diagnosis.

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 that ability of sensors and effectors of *both* meta-levels to access the same object-level is advantageous (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  $M_2$ . This problem is already well known in distributed fault-tolerance. We will return to it 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. However, they may mutually support each other's learning, instead of inhibiting each other.

### 5.1.4 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.

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

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.

## 5.2 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, while  $M_1$  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),  $M_2$  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. [5].)

How the ontology is *populated* will depend on whether  $M_2$  is reasoning about  $M_1$  or about  $O_1$ , since the actual values of the content will be different in the two cases: the first involves rule firing traces about different things (the external world or other internal traces), the second is the trace of attention focus on different things. This leads us back to re-use of the same components for different roles mentioned earlier.

## 5.3 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) or where there is a shared domain. 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.

Additional coordination mechanisms are required for global explanation and action. Many of the techniques that apply to multi-agent systems and fault-tolerance can be applied here. In situations where there is only one correct decision, a classic fault-tolerance approach is to use a majority voting system, which needs at least three components. The coordination of such a system requires distributed consensus (e.g. the majority have to agree about the result of the vote). This is a non-trivial problem in fault-tolerance if one or more components can disrupt the voting process (e.g. by "lying") but significant progress has been made here (see e.g. [19] for an overview). If there is only a requirement to make a decision eventually, the first meta-level to provide evidence of why it believes the other one is wrong could be allowed to "win" the competition and inhibit the other meta-level(s).

Coordination means that more complexity has to be added to the meta-levels, which makes them in turn more difficult to be monitored. In [8] 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, but as part of a "training simulation". Although this worked successfully in a restricted artificial environment, it remains a significant challenge for large real-world systems.

### 5.3.1 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 component (possibly another meta-level) that collects information from all 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 [1] although the content of the global workspace may be changing during the course of the explanation. The implementation in [16] includes a “meta-managerial” critic, which has a similar global function.

To satisfy the requirement of closed meta-levels (where all meta-levels are monitored), a meta-level which is constructing the explanation must itself be subject to critical evaluation in the same way as other meta-levels. 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. [11]).

## 6 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

- [1] Bernard J. Baars. *A Cognitive Theory of Consciousness*. New York: Cambridge University Press, 1988.
- [2] Luc P. Beaudoin. *Goal Processing in Autonomous Agents*. PhD thesis, University of Birmingham, 1994.
- [3] Michael T. Cox. Metareasoning, monitoring, and self-explanation. In *Proceedings of the First International Workshop on Metareasoning in Agent-based Systems at AAMAS-07*, pages 46–60, 2007.
- [4] Michael T. Cox and Anita Raja. Metareasoning: A manifesto. Technical Report BBN Technical Memo, BBN TM 2028, BBN Technologies, Intelligent Computing, Cambridge, MA 02138, 2007.
- [5] Roberta Ferrario and Alessandro Oltramari. Towards a Computational Ontology of Mind. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS 2004)*, pages 287–297. IOS Press Amsterdam, November 2004.
- [6] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.
- [7] S. A. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation*, 8(4):443–473, 2000.
- [8] C. M. Kennedy. *Distributed Reflective Architectures for Anomaly Detection and Autonomous Recovery*. PhD thesis, University Of Birmingham, Birmingham, UK, July 2003.
- [9] C. M. Kennedy and A. Sloman. Acquiring a Self-Model to Enable Autonomous Recovery from Faults and Intrusions. *Journal of Intelligent Systems*, 12(1), 2002.

- [10] C. M. Kennedy and A. Sloman. Autonomous Recovery from Hostile Code Insertion using Distributed Reflection. *Journal of Cognitive Systems Research*, 4(2):89–117, 2003.
- [11] Christof Koch. *The Quest for Consciousness: A Neurobiological Approach*. Englewood, Colorado: Roberts and Company Publishers, 2004.
- [12] Marvin Minsky, Push Singh, and Aaron Sloman. The St. Thomas common sense symposium: designing architectures for human-level intelligence. *AI Magazine*, 25(2):113–124, 2004.
- [13] Melanie Mitchell. Self-Awareness and Control in Decentralized Systems. In *Working Papers of the AAAI 2005 Spring Symposium on Metacognition in Computation*, Menlo Park, CA, USA, 2005.
- [14] Linda M. Northrop. SEI’s Software Product Line Tenets. *IEEE Software*, 19(4):32–40, 2002.
- [15] Anita Raja and Victor Lesser. A Framework for Meta-level Control in Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 15(2):147–196, 2007.
- [16] Push Singh. *EM-ONE: An Architecture for Reflective Commonsense Thinking*. PhD thesis, Artificial Intelligence Lab, MIT, 2005.
- [17] A. Sloman and R. Poli. SimAgent: 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.
- [18] Aaron Sloman. Varieties of Affect and the CogAff Architecture Schema. In *Symposium on Emotion, Cognition, and Affective Computing at the AISB’01 Convention*, March 2001.
- [19] Paulo Verissimo and Luis Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.