Decentralised Metacognition in Context-Aware Autonomic Systems: some Key Challenges

Catriona M. Kennedy
MIT Computer Science and AI Lab,
Cambridge MA 02139
cmk@csail.mit.edu

Abstract

A distributed non-hierarchical metacognitive architecture is one in which all meta-level reasoning components are subject to meta-level monitoring and management by other components. Such metacognitive distribution can support the robustness of distributed IT systems in which humans and artificial agents are participants. However, robust metacognition also needs to be context-aware and use diversity in its reasoning and analysis methods. Both these requirements mean that an agent evaluates its reasoning within a "bigger picture" and that it can monitor this global picture from multiple perspectives. In particular, social context-awareness involves understanding the goals and concerns of users and organisations.

In this paper, we first present a conceptual architecture for distributed metacognition with context-awareness and diversity. We then consider the challenges of applying this architecture to autonomic management systems in scenarios where agents must collectively diagnose and respond to errors and intrusions. Such autonomic systems need rich semantic knowledge and diverse data sources in order to provide the necessary context for their metacognitive evaluations and decisions.

1. Introduction

Real world distributed IT systems may be regarded as a composition of artificial and human agents (socio-technical systems). Autonomic management (Sterritt et al. 2005) of such systems will become increasingly necessary, since frequent manual configuration and fault-diagnosis is error-prone and not feasible in the longer term.

Robust autonomic management requires the managing agents to monitor and manage all aspects of the system, including their own reasoning and control decisions. The ability of an agent to reason about its reasoning is called "metareasoning" (Cox and Raja 2007) or more generally metacognition. The agents may also support human metacognition by alerting them to errors or misunderstandings while they use the system. This is particularly required in safety-critical applications (such as for example disaster management) but is also important in other areas such as collaborative research or learning.

We identify the following three requirements for robustness in socio-technical systems:

1. Distribution and decentralisation of all aspects including the metacognition. This means that all kinds of reasoning should be subject to questioning and criticism (provided that coordination mechanisms are used).
2. Diversity of methods: different and independent ways to think about things and to do things (Minsky 2006).
3. Social context-awareness: understanding goals and concerns of users and organisations, social norms etc;

In earlier work (Kennedy 2008) we argued for decentralised metacognition to resist subversion of meta-level reasoning by hostile code. A decentralised architecture is one in which all meta-level reasoning components are subject to meta-level monitoring and management by other components (the first requirement for robustness above). In this paper we extend this work to address the need for diversity and for social context-awareness. Both these requirements relate to the need for a metacognitive agent to evaluate its reasoning within a "bigger picture" and to see this larger picture from multiple perspectives (using different conceptualisations and analysis methods).

The paper is structured as follows: the first part presents a conceptual architecture for metacognition with general (non-social) context-awareness and diversity in a simple robotics domain. The second part considers its application to a socio-technical domain in which autonomic management agents are aware of user concerns and goals. Finally we discuss some key challenges and propose ways to address them.

2. Role of Context in Metacognition

Figure 1 shows a schematic diagram of metacognition which emphasises the role of context and the need for different kinds of meta-level control. Figure 1 (a) shows an agent as a two-layer structure containing an object-level O_1 and a meta-level M_1 respectively. The sensors and actuators (ground level) are not shown for simplicity.

The monitoring function (labelled m) of M_1 requires a reasoning trace T_1 of O_1. A trace can include information on active goals, assumptions made, conclusions drawn and knowledge sources used (among other things) by a reasoning process (Morgan 2009). The meta-level component M_1

(which we call an M-box) is a software component which analyses the trace to check if the object-level O_1 (O-box) satisfies its requirements (such as making progress towards a goal). M_1 also attempts to debug any problems. The control arrow (c) modifies O_1 if the M-box has identified a bug that can be corrected. An M-box can contain various anomaly-

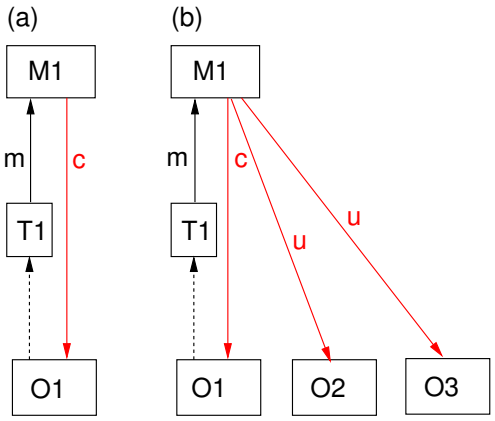

Figure 1: Meta-level monitoring and control: (a) simple control loop; (b) Agent activates other object-level methods to compare with O_1

detection and performance evaluation tests. Examples include case-based reasoning (Singh 2005) to test for known faults.

In many realistic scenarios, an agent must evaluate its reasoning with regard to the current situation in the environment, and particularly whether its reasoning or knowledge is appropriate when a new situation arises. Since it would be illogical to trust the same object-level method that it is debugging, it needs to have *alternative* object-level methods available which it uses to gather evidence about the context.

For example, if a robot is expecting an office at the end of a corridor, and finds a cupboard instead, it may think it has encountered a very small office (since it knows nothing about cupboards). When its meta-level analyses the reasoning trace, it can consider two possibilities: (a) the object-level has made an error (such as in navigation or in feature recognition) and it is not really at the end of the corridor it thinks it is in or (b) the object-level knowledge of the building is incomplete, which led to a wrong expectation about this particular corridor. To rule out (a), it can explore the area using alternative object-level methods (for route planning, sensing and perception). If they all give the same anomalous result, the robot may conclude instead that its knowledge is incomplete and plan to learn a more accurate map of the building.

2.1 Role of Diversity

Figure 1(b) shows the use of additional object-level methods to collect independent evidence about the environment. These methods are trusted by the agent at that point (arrows labeled "u") and may also be used to probe whether an anomalous result was only reported by O_1 or whether the other methods produce similar results. Therefore M_1's usage of O_2 and O_3 help it to monitor and debug O_1. The activation and suppression of different methods may happen dynamically as in the critic-selector architecture of (Minsky 2006). Similarly the level of detail of M_1's monitoring may be varied dynamically in response to problems.

Two different kinds of control are illustrated here. One kind of control acts on the object-component it is debugging, for example to modify or suppress it as a result of fault diagnosis. Another kind of control activates alternative object-level methods.

2.2 Distributed metacognition

A meta-component (M-box) may contain errors or its monitoring may be unnecessary and cause inefficiency. In distributed metacognition, several M-boxes participate in the monitoring and control of the system so that all M-boxes are monitored.

In previous work (Kennedy and Sloman 2003) we implemented distributed metacognition for a simulated autonomous vehicle using the *SimAgent* toolkit (Sloman and Poli 1995). This implementation detects anomalies as deviations from learned models of normal rule firing. In addition to dangers in the virtual world, hostile agents can attack the O- or M-boxes controlling the vehicle, including those detecting and repairing errors. Self-repair is implemented as automated patching of corrupted rule-systems. Repair decisions are made by majority voting among replicated components, on the assumption that these are not all attacked simultaneously.

Introducing diversity Replication has the disadvantage that a vulnerability or design error may also be replicated (or exploited repeatedly). To overcome this problem, we propose to introduce diversity by using different meta-level methods or "perspectives". An example configuration show-

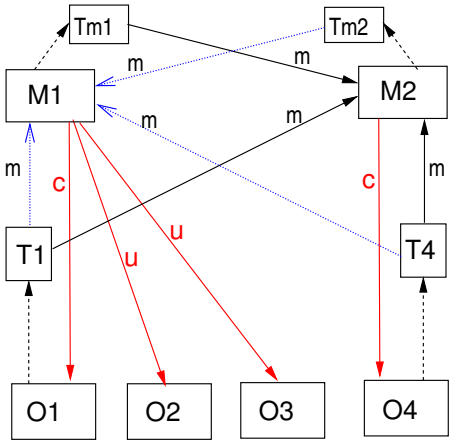

Figure 2: A snapshot of distributed metacognition

ing perspectives is shown in Figure 2. The dashed arrows indicate that reasoning events are recorded in a trace. For the ith O-box, T_i is the trace of O_i and similarly for the M-boxes Tm_i is the trace of M_i. The fuzzy dotted arrows (pointing

towards M_1) show the perspective of M_1, while the black filled arrows show the perspective of M_2.

Figure 2 shows a configuration in which metacognition is distributed in two ways:

1. M-boxes monitor and evaluate O-boxes in different ways. E.g. one monitors traces of reasoning processes (such as matching rules), another monitors changes in memory contents. Different methods and criteria of evaluation may also be used. These represent the different perspectives.

2. M-boxes also monitor each other. In some cases they may also activate or suppress each other, provided such meta-level control actions are coordinated (for example, if there is sufficient agreement among M-boxes before taking an action).

The advantage of using the second kind of distribution in addition to the first is that it makes it possible to debug an M-box and explain the *cause* of the error in situations where the M-box is at fault, thus enabling self-explanation (Cox 2007) in those situations.

Context-aware debugging of meta-level reasoning In the same way as for the debugging of an O-box, an M-box's reasoning needs to be analysed within a context. In addition to events in the world, the meta-level context includes the current reasoning processes that are active at the time. For example, M_2 may need to know how M_1 is reasoning when certain object-level reasoning events happen (such as forgetting something or when making a decision).

In a purely hierarchical system with two layers of M-boxes, we would normally expect M_2 to have a separate role from M_1. M_2's role may be called the "meta-meta"-role in contrast to the "meta-object"-role of M_1. Therefore the upper layer (M_2) would need to trust alternative meta-level methods at the lower layer (playing the "meta-object" role) to deliver the "current state" of the agent's object-level reasoning. However, a hierarchical system involves a proliferation of components which cannot all be monitored, thus making the system fragile. Therefore we propose a non-hierarchical architecture.

Reusablity of meta-level reasoning methods A requirement for such a non-hierarchical closed architecture is that the same component must be able to play both the roles of "meta-meta" and "meta-object" (MM and MO respectively). Therefore M_2 can directly analyse the trace of O_1 (without intermediaries) provided that its method of analysing M_1's trace will also suffice for O_1. In this way, M_2 is providing an alternative view of O_1, which may be different from that of M_1.

For this requirement to be met, an object-level trace should not require fundamentally different processing from a meta-level trace (although differences in detail can exist). For example, M_2 may detect that a particular goal has been forgotten in either O_1 or M_1. The concepts of reusablility and polymorphism in software engineering are important here.

2.3 Social metacognition

In traditional multi-agent systems as in human societies, components are independent agents with no direct access to each other's reasoning traces, but may instead infer reasoning traces by observing spoken communication or by accessing a shared log showing the history of a debate or a series of decisions.

Similarly, agents cannot activate or suppress each other directly but must use accepted social rules and forms of communication. The agents must agree to cooperate to solve a given problem. Although they may disagree about specific subgoals, they have a common goal to solve the disagreements.

Figure 2 can be translated into a social (multi-agent) system if the connections are changed as follows:

- M_1 and M_2 become M-Boxes for separate agents, with separate O-Boxes. For example M_1 together with O_1, O_2 and O_3 become an agent A1 while M_2 and O_4 become A2.
- Diverse object-level methods (O-Boxes): in addition to activating its own alternative O-Boxes directly, an agent may ask for another agent's opinion on its external behaviour or about the state of the environment. Particularly, critical evaluation from the viewpoint of the other agent is important.
- Diverse meta-level methods (M-Boxes): it may be feasible for agents to ask each others' opinions on the correctness of their reasoning, assuming that their reasoning traces can be made available for inspection.
- Distributed control: Instead of M-Boxes activating or suppressing each other, agents are willing to be corrected by other agents.

A social metacognitive architecture may involve individual agents with distributed metacognition which are in turn embedded within a cooperative multi-agent system.

Humans as participating agents Humans may be "components" in such a distributed metacognitive system. If we assume a computational model of human cognition (and metacognition), a human (or group of humans) may be the "agent" whose reasoning is being critically evaluated by another agent (itself software or human) without fundamentally changing the architecture. The converse of this is traditional debugging where humans monitor agent reasoning. We can list the different forms of social metacognition as follows:

1. A-A: artificial multi-agent metacognition or "social" metacognition within one agent.
2. A-H: an agent analyses a human reasoning trace.
3. H-A: traditional debugging.
4. H-H: human social metacognition (also includes psychology and social sciences).

Figure 2 may be translated into a multi-agent (A-A) scenario or a human-agent (A-H) scenario. If some form of human reasoning trace is available, many of the same debugging

strategies may be applicable to both A-A and A-H. For example, a human may fail to use relevant knowledge in the same way as an artificial reasoning process. Such "debugging" would typically be an assistive process to help human metacognition. Intelligent tutoring systems (Bull and Kay 2008) can do this by showing users the system-generated model of the user's learning history.

3. Autonomic and Self-Adaptive Systems

We will now consider whether the above concepts of social metacognition can be applied in an autonomic computing scenario. The objective of autonomic computing or "self-management" is to automate many aspects of system administration, such as reconfiguration, fault-diagnosis, protection from security violations and optimisation (Sterritt et al. 2005). Similarly self-adaptive software (Salehie and Tahvildari 2009; Robertson and Williams 2006) should have the capability to adapt itself to ongoing changes with less necessity for system administrators and programmers to make manual updates.

3.1 Autonomic Control Loop

An autonomic agent has been defined as a control loop known as MAPE-K (Monitor Analyse Plan Execute + Knowledge) by (Sterritt et al. 2005). We divide this loop into two components:

1. *Monitoring and analysis*: e.g. anomaly-detection, performance-evaluation, fault diagnosis;
2. *Planning and execution*: e.g. resource-allocation, configuration and coordination; planning and action is done in response to specified goals and policy but may also be *corrective* as a result of problems that were detected by monitoring.

Both are mutually interdependent. Monitoring depends on resource-allocation, since it needs sensor coverage as well as processing power for timely analysis of sensor data and planning of corrective action. Conversely, the effectiveness of control (including resource management) depends on monitoring of its performance and timely detection of problems.

The following are examples of managed systems:

- systems for accessing data (e.g. database servers, semantic web portals).
- software tools and middleware that need to be adapted;
- networks
- data-center and "cloud" resources.

We include self-adaptive software within our definition of autonomic systems, since they are an important subclass of "self*" systems (self-configuration, self-protection etc). An autonomic system with self-adaptive software would not only monitor and reconfigure the managed system but also its own software. Instead of policies which would be used to manage devices and computing resources, self-adaptive software would normally have a representation of the software requirements at run-time. Its self-monitoring and analysis would involve "self-debugging", which may be regarded as a generalisation of self-healing and self-protection, if we define a "bug" as an error or deliberately caused fault (such as an intrusion).

Importance of ontologies and models Autonomic management agents require a language and ontology for specifying models of the managed system and its desired behaviour. This is necessary in order to reason about possible future states of the managed system and to predict faults. Ontologies are also needed to define objects and the possible actions on them so that re-configurations may be planned. Recent work on ontologies includes the description and modelling of networked devices (Serrano, Serrat, and Strassner 2007) and the FOCALE adaptive autonomic architecture (Strassner, Agoulmine, and Lehtihet 2006).

3.2 Metacognition in Autonomic Systems

A meta-level for an autonomic agent would effectively be a second MAPE-K loop which monitors and controls the object-level MAPE-K loop. It may reason about any of the following:

1. Monitoring and analysis methods: e.g. meta-level determines what data should be collected about the managed infrastructure and social environment and how often.
2. Planning and execution: e.g. meta-level monitors and controls the planning and execution of reconfigurations, or responses to intrusions.
3. Current knowledge and goals: e.g. meta-level critically evaluates and adapts the models and policies currently being applied by the agent.

3.3 Context Aware Autonomic Management

Diagnosis of a problem needs context-awareness. Effectively we need an autonomic system to understand and adapt to its environment in a similar way to a robot (although the environment here is very different). We define an autonomic system as context-aware if it is aware of the *semantic content* of at least some of the applications it is managing and it can adapt its management dynamically in response to new developments within the semantic content. This is sometimes known as adaptive infrastructure. An example is the LEAD project (Plale et al. 2005) which adapts its processing and sensor coverage in response to emerging weather events.

Autonomic management can also involve the planning of action sequences (scripts) for reconfiguring software components (Srivastava, Bigus, and Schlosnagle 2004). In a non-context-aware system, the reconfiguration has no knowledge of what the software is used for. Context-awareness involves some understanding of this purpose. Ideally the system is goal-directed and the managed software contains a solution to a problem that the system has found by reasoning about an application domain.

As an example of context-aware meta-level control, we can imagine an autonomic monitoring agent that uses a model or simulation of a managed system (e.g. a sensor network) to predict its future states and to detect problems (Williams et al. 2004; Robertson and Williams 2006). A discrepancy is detected between the model-predicted state

and the observed state. The measured state also seems much "worse" than the model-predicted state, when considered in the context of a critical application (e.g. severe weather tracking). Therefore an urgent re-allocation of computing resources is required in order to investigate and diagnose the problem. The problem may be in the network itself or the autonomic monitoring and analysis algorithms may be faulty.

3.4 Towards Social Context Awareness

We can define the system as *socially context-aware* if it is aware of *user concerns* and *goals* and acts on behalf of them. For example, if the users are experimental physicists, they are concerned about the results of experiments and they want to test hypotheses; if they are emergency health workers, they are concerned about diagnoses and critical life support monitoring. Recognition of user goals is addressed in (Smith and Lieberman 2010).

The social environment is the context in which events in an IT infrastructure take place. For example, file access patterns and network activity can be associated with user goals and actions, which can in turn be associated with organisational goals and wider social networks. Determining causality is also feasible; a network intrusion event may be traced to a social event which is a violation of a social norm (e.g. fraud).

The interaction between IT infrastructure and social environment can be represented as different levels shown in Figure 3. This diagram shows multiple levels at which a socio-technical system can be modelled and observed.

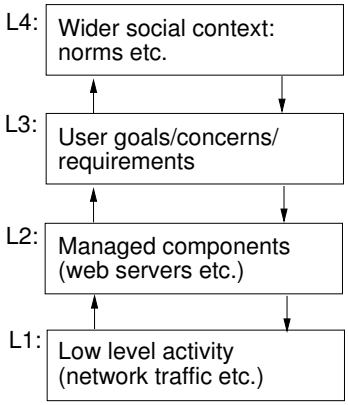

Figure 3: Multiple levels of an IT system within a social context

Models and data sources This kind of deep context awareness ideally needs commonsense knowledge about the world and about social norms. Steps in this direction are possible using agent-based social models (Epstein 2006), where an "agent" can represent an artificial or human agent. These models might be developed from existing theories or directly from observation and learning. Some may be classed as meta-level models and may be developed by observing reasoning and decision processes. Such models may later be used for meta-level reasoning (for example, to detect typical reasoning flaws) or to reason about the limits or relevance of the knowledge that is being taken into consideration.

These different kinds of models and their associated data (from which the models may be built or adapted) include the following:

- Models of the infrastructure - devices, networks, software, applications etc.
- Models of the managing agents. These would be partial models, each corresponding to a perspective held by an agent.
- User models (cognitive models and goals).
- Social models - generated or updated using various data sources such as transaction logs, text etc.

Figure 4 shows a schematic diagram of agents managing and modelling an IT infrastructure. Many different social models may be constructed from related data sources.

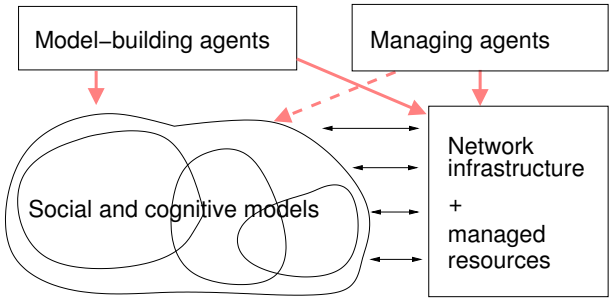

Figure 4: Social context awareness

Data fusion and high-level descriptions Just as in the robotics example, components (agents) may cooperate together to build global or partial pictures from different perspectives. A perspective (held by an agent) may represent a conceptual categorisation (an ontology) which is used to interpret and fuse data from multiple sources. Such perspectives might represent the different concepts and priorities of the various users in an organisation. Multiple agents can interpret the same data in different ways according to their high level conceptualisation. They may also use specialist data sources. In a configuration such as Figure 2, each agent might have its own specific algorithms (O-Boxes).

Simulations based on those ontologies may be used on the different levels of abstraction in Figure 3 and may be continually updated from different data streams. For example, a simulation on L4 may use data streams from several different sources on the lower levels (e.g. software usage, conversations, network traffic etc). The simulations generate ongoing expectations which also guide the data collection (or "attention focus") of the agents. More details of this general concept are in previous work (Kennedy et al. 2007).

Automated or (semi-automated) building of such models and their subsequent refinement and testing is a non-trivial meta-level control operation. Potential policy conflicts (for

example, involving privacy) would also need to be resolved (Lupu and Sloman 1999).

4. Towards Distributed Metacognition in Autonomic Systems

In this section we address the challenges of applying the architecture in Figure 2 to the above socio-technical scenarios and suggest directions for solving them. We identify some key requirements below:

1. Automated diagnosis and correction: cooperative agents need to evaluate and correct each other, while taking into account the wider context. This requires solution of the following sub-problems:
   (a) Limiting complexity: aiming for reusability of methods for analysing different reasoning traces.
   (b) Mutual evaluation (criticism) requires translation across different representations held by each agent.
   (c) Mutual adaptation: not only is a translation between perspectives required, but also a method of adaptation to each other's requirement, with possible learning of new concepts.
2. Subversion resistance: multiple agents need to agree to overrule a hostile agent. A common understanding of "acceptable goals" may be necessary.

4.1 Automated Diagnosis and Debugging

Context-aware error detection and correction depends on the perspective of the monitoring agent. If an agent represents a perspective which is defined according to user concepts and goals then its "debugging" method may include the evaluation of whether those goals are being satisfied, and whether the relevant concepts are being taken into account.

For example, an agent A1 representing the concern of information integrity (such as in medical imaging) might evaluate its own (object-level) reasoning according to how well it meets integrity requirements in changing contexts. A1 might also apply the same evaluation process to another agent A2 responsible for applying an energy management policy (in accordance with the first type of distributed metacognition in Figure 2). A "bug" in A2's object-level reasoning would be a failure of A2 to ensure that the energy conservation does not violate information integrity. Therefore it needs to be aware of these constraints.

Reusability A1 may also detect a bug in A2's *meta-level* reasoning. For example, it may determine whether A2's meta-level's considerations and goals contradict the information integrity requirement. Since we wish to reuse as far as possible the evaluation method for both meta-level and object-level, a generic method of sending a negative evaluation may be feasible, although the interface with the different traces would not be identical.

Translation across representations Agents can send each other alerts and correction messages if they are not satisfying each other's requirements. However, if agents use different sets of concepts and representations then a translation is necessary between perspectives.

The binding problem in cognitive science addresses this kind of translation when applied between sensory modalities. One solution to the binding problem is in (Hawes et al. 2007). If we imagine that one "agent" is responsible for processing visual input and another for auditory, translation between these modalities allows, for example, the auditory agent to alert the vision agent to focus on a human who has just begun speaking. A similar translation is possible between events on the different levels of abstraction in Figure 3 and between the different models in Figure 4.

Mutual adaptation In the visual-auditory example above, agents can give feedback to each other to focus attention on certain stimuli. If they have access to each other's reasoning traces, they can monitor focus of attention and send corrective alerts as necessary (e.g. not giving sufficient attention to relevant events). In response to a correction alert, the vision agent can learn to recognise the visual cues when a person speaks by correlating alerts it receives from the auditory agent with new visual features in the environment. Effectively agents help to "debug" each other interactively.

An alert from another agent would be similar to an unexpected event in the environment. The agent would respond by planning to learn (Cox and Ram 1999). Subsequent monitoring and conflict resolution in the learning process would also be important (Kim and Gil 2008). If necessary, the vision agent can learn a new concept ("person speaking") which it can add to its ontology. (Afsharchi and Far 2006) addresses this kind of problem in a more general multi-agent context.

4.2 Subversion Resistance

If a security monitoring agent is subverted, its meta-level control may activate hostile code, causing it to send a false alert stating that a security problem has been detected. If other agents find no evidence of an intrusion, this does not prove that there is no problem. Therefore, resources are allocated to its detection, possibly diverting resources away from a real problem. If agents can identify a bug in the agent's meta-level control, they can explain the cause of the problem and disable the rogue code.

Benign agents must recognise the potential for hostility and agree on joint action. Recognising hostility requires social context-awareness as well as diverse and independent ways of gathering evidence. In particular, metareasoning about human reasoning traces is useful in this context.

Agreement on whether an agent is potentially damaging requires that the agents have a common understanding of what kind of activity is acceptable or necessary and what kind of activity is hostile. One possible approach is to define goals on the meta-level that agents agree to pursue. These are requirements to be satisfied by the management agents' reasoning processes. Such goals can include the following:

1. to learn about other agents' requirements and goals (including user goals);
2. to cooperate with other agents' object-level goals, provided that:
   (a) these goals are consistent with agreed meta-level goals;

(b) if there are conflicts between object-level goals, they generate a new goal to try to overcome these specific conflicts;

Agents may have different representations of these requirements, but they should be able to test if an observed agent's activity and reasoning trace satisfies them. A "friendly" agent (executing non-hostile code) which is merely in error should not contradict these requirements. In other words, it must be seen to make an effort to cooperate by a sufficient number of agents, which are monitoring the agent in different ways. Research on trust and reputation systems is relevant for this problem (Jøsang, Ismail, and Boyd 2007) as well as agent-based modelling (Epstein 2006).

5. Related Work

Research on multi-agent metacognition such as (Raja and Lesser 2007) is addressing many of the coordination problems that a distributed metacognitive system will have to solve. Although the mult-agent approaches do not involve the kind of distribution of metacognition we defined here, they have to address similar issues, such as connectivity between agents and the resources that should be allocated to metacognition. Planning to learn (and deciding what and how to learn) in response to anomalies is also an important requirement for context-aware metacognition. Relevant work on this includes (Cox and Ram 1999; Josyula et al. 2009; Kim and Gil 2008).

We have considered an autonomic system as a cognitive system. A cognitive architecture for autonomic systems is proposed in (Kramer and Magee 2007), which involves online planning and deliberation with continual self-configuration in response to changes. Other approaches are based on multi-agent systems. For example (Das et al. 2008) uses agents for power management. Autonomic context-awareness is currently being addressed in simple scenarios (Klein et al. 2008; Sitou and Spanfelner 2007). An example of metareasoning in context-aware self-adaptive software is (Robertson and Laddaga 2009), where a metareasoner evaluates the accuracy of predictions made by vehicle trackers (reasoners) and selects the best performing ones for further refinement.

Mutual debugging among agents assumes that conflicts between requirements or policies can be resolved. A common approach to conflict resolution is to use a "meta-policy" giving guidelines or constraints on the application of policy rules (Lupu and Sloman 1999). For example, precedence or priority of one rule over another may be specified in a meta-policy (Kagal, Finin, and Joshi 2003).

6. Summary and Conclusion

In this paper we showed how context-awareness and diversity can be incorporated into distributed metacognition. We then argued that these principles can also be applied to agents in an autonomic management system. Automated recovery in such a system means that the agents have to participate in monitoring and debugging each other, since there is no hierarchical system of evaluation.

The following are challenges for automated recovery:

- If agents are cooperative, they can share reasoning traces and help each other learn their respective requirements. Translation between representations is necessary.
- If one agent is hostile because its meta-level has been subverted, it may not respond to error correction requests. In this case, agents have to agree to overrule the hostile agent. Social context awareness is important here to identify the causes and motivation of an attack.
- An important software engineering requirement is that the meta-level methods for analysing traces are reusable and polymorphic as far as possible, so that they can be applied to object-level traces as well as meta-level traces with minimal changes. Otherwise, the complexity of additional meta-level components can make the system fragile.

Acknowlegement

This work was funded in part by the Defense Advanced Research Projects Agency.

References

Afsharchi, M., and Far, B. H. 2006. Automated ontology evolution in a multi-agent system. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, 16. New York, NY, USA: ACM.

Bull, S., and Kay, J. 2008. Metacognition and Open Learner Models. In *3rd Workshop on Meta-Cognition and Self-Regulated Learning in Educational Technologies, at the 9th International Conference on Intelligent Tutoring Systems (ITS2008)*.

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

Cox, M. T., and Ram, A. 1999. Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence* 112:1–55.

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.

Das, R.; Kephart, J. O.; Lefurgy, C.; Tesauro, G.; Levine, D. W.; and Chan, H. 2008. Autonomic Multi-agent Management of Power and Performance in Data Centers. In *AAMAS-08: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-agent Systems*, 107–114. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Epstein, J. M. 2006. *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press (Princeton Studies in Complexity).

Hawes, N.; Sloman, A.; Wyatt, J.; Zillich, M.; Jacobsson, H.; Kruijff, G.-J.; Brenner, M.; Berginc, G.; and Skočaj, D. 2007. Towards an Integrated Robot with Multiple Cognitive Functions. In Holte, R. C., and Howe, A., eds., *Proceedings of the Twenty-Second AAAI Conference on Arti-*

ficial Intelligence (AAAI 2008), 1548 – 1553. Vancouver, Canada: AAAI Press.

Jøsang, A.; Ismail, R.; and Boyd, C. 2007. A Survey of Trust and Reputation Systems for Online Service Provision. *Decision Support Systems* 43(2):618–644.

Josyula, D. P.; Hughes, F. C.; Vadali, H.; Donahue, B. J.; Molla, F.; Snowden, M.; Miles, J.; Kamara, A.; and Maduka, C. 2009. Metacognition for Self-Regulated Learning in a Dynamic Environment. In *Procceedings of the Workshop on Metareasoning in Self-Adaptive Systems at the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, (SASO 2009)*.

Kagal, L.; Finin, T.; and Joshi, A. 2003. A Policy Language for A Pervasive Computing Environment. In *Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks*.

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.; Theodoropoulos, G.; Sorge, V.; Ferrari, E.; Lee, P.; and Skelcher, C. 2007. AIMSS: An Architecture for Data Driven Simulations in the Social Sciences. In *Workshop on Dynamic Data-Driven Applications Simulation at ICCS 2007*. Beijing, China: Springer-Verlag.

Kennedy, C. M. 2008. Distributed Meta-Management for Self-Protection and Self-Explanation. In *Proceedings of the AAAI-08 Workshop on Metareasoning: Thinking about Thinking*.

Kim, J., and Gil, Y. 2008. Developing a Meta-Level Problem Solver for Integrated Learners. In *Proceedings of the AAAI-08 Workshop on Metareasoning: Thinking about Thinking*.

Klein, C.; Schmid, R.; Leuxner, C.; Sitou, W.; and Spanfelner, B. 2008. A survey of context-adaptation in autonomic computing. In *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems*, 106–111. IEEE Computer Society.

Kramer, J., and Magee, J. 2007. Self-managed systems: an architectural challenge. In *FOSE '07: 2007 Future of Software Engineering*, 259–268. Washington, DC, USA: IEEE Computer Society.

Lupu, E. C., and Sloman, M. 1999. Conflicts in policy-based distributed systems management. *IEEE Trans. Softw. Eng.* 25(6):852–869.

Minsky, M. 2006. *The Emotion Machine*. New York: Simon and Schuster.

Morgan, B. 2009. Funk2: A Distributed Processing Language for Reflective Tracing of a Large Critic-Selector Cognitive Architecture. In *Workshop on Metareasoning in Self Adaptive Systems at the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2009)*.

Plale, B.; Gannon, D.; Reed, D.; Graves, S.; Droegemeier, K.; Wilhelmson, B.; and Ramamurthy, M. 2005. Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD. In *Workshop on Dynamic Data Driven Application Systems at the International Conference on Computational Science (ICCS 2005)*.

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.

Robertson, P., and Laddaga, R. 2009. Metareasoning Based Self Adaptive Tracking. In *Workshop on Metareasoning in Self Adaptive Systems at the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2009)*.

Robertson, P., and Williams, B. C. 2006. Automatic Recovery from Software Failure: A Model-based Approach to Self-Adaptive Software. *Communications of the ACM* 49(3):41–47.

Salehie, M., and Tahvildari, L. 2009. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous Adaptive Systems* 4(2):1–42.

Serrano, J.; Serrat, J.; and Strassner, J. 2007. Ontology-Based Reasoning for Supporting Context-Aware Services on Autonomic Networks. In *Proceedings of the IEEE International Conference on Communications (ICC-07)*, 2097–2102.

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

Sitou, W., and Spanfelner, B. 2007. Towards requirements engineering for context adaptive systems. In *Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 2*, 593–600. IEEE Computer Society.

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.

Smith, D., and Lieberman, H. 2010. The Why UI: Using Goal Networks to Improve User Interfaces. In *Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI-2010)*.

Srivastava, B.; Bigus, J.; and Schlosnagle, D. 2004. Bringing Planning to Autonomic Applications with ABLE. In *International Conference on Autonomic Computing (ICAC 2004)*, 154–161.

Sterritt, R.; Parashar, M.; Tianfield, H.; and Unland, R. 2005. A Concise Introduction to Autonomic Computing. *Advanced Engineering Informatics* 19(3):181 – 187.

Strassner, J.; Agoulmine, N.; and Lehtihet, E. 2006. FOCALE: A Novel Autonomic Networking Architecture. In *Proceedings of the Latin American Autonomic Computing Symposium (LAACS-2006)*.

Williams, B. C.; Ingham, M.; Chung, S.; Elliott, P.; and Hofbaur, M. 2004. Model-based Programming of Fault-Aware Systems. *AI Magazine* 24(4):61–75.