A General Architecture for Flexible Autonomous Systems

Louise Dennis, Elisa Cucco, Michael Fisher
Department of Computer Science, University of Liverpool, UK

Abstract

We describe an architectural approach to autonomous systems that not only provides generality and verifiability, but also *flexibility* through self-awareness and self-reconfiguration. This is an agent-based approach where the core *rational agent* oversees a range of feedback control systems (within a modular architecture) handling adaptive interaction with the system's environment. The rational agent makes all high-level decisions and, crucially, is able to provide *reasons* for its choices. In addition, the agent is self-aware being able to assess activity within the system's modular architecture, and is able to reconfigure this architecture to cope with changes, either in its hardware or in the environment.

This architectural approach has been instantiated in a number of application areas, including *autonomous satellites*, *convoys of driver-less cars*, *unmanned air vehicles*, and *autonomous nuclear decommissioning*. In several of these, notably the first and last, the flexibility and resilience provided by self-reconfigurability is crucial.

1 Introduction

There are a number of different implementation routes for autonomous systems, and components within these are often categorised as *symbolic* or *sub-symbolic*. Symbolic systems have clear descriptions and semantics, typically involving logic-based or declarative approaches. Sub-symbolic systems are more complex with behaviour being dependent on feedback loops involving environmental interaction. Coarsely, the latter tend to be efficient but opaque while the former tend to be less efficient but more transparent. Typical examples of the latter are *adaptive control systems*, *neural networks*, *genetic algorithms*, *reinforcement learning*, etc.

Although, in academic circles, attempts have been made to produce purely symbolic or purely sub-symbolic architectures for autonomous systems, neither extreme is particularly satisfactory. In practice, autonomous systems architectures involve a combination of symbolic and sub-symbolic components. The closest to one of these extremes are *hierarchical feedback control systems*, such as Brooks' *subsumption architecture*. However, with the increasing requirements of verifiability, transparency, and flexibility, such architectures appear rare.

Autonomous systems are increasingly popular, being developed for, and deployed in, a wide range of scenarios. Here, we are particularly concerned with applications that *essentially* require autonomy such as working environments that are dangerous and hazardous for humans. In such cases, any human involvement must at least be remote and often very limited. A second class of applications is *not* so extreme but typically involves an industrial environment where a human operator, monitoring the activity, might need to be: physically remote from the working area; control a large number of robots; or perform very repetitive tasks. In this case, the mundane and repetitive nature of the work can lead to the operator becoming distracted, disinterested, and even taking undue risks [Dennis *et al.*, 2014]. In both these scenarios, we would clearly benefit from the autonomous (robotic) system having greater autonomy and being able to deal with key decisions and problems on its own.

The basic tasks, within typical mission goals, include deciding what is the appropriate action, what is the appropriate object to be acted on, what is the appropriate order of action executions, and what is the appropriate way of performing every action, without the human intervention. Increasingly important for truly autonomous systems is the need to be reflexive, to have a clear idea of the system components and their efficacy, and self-reconfigurable, to be able to dynamically modify its own software[1] organisation.

To deal with all these issues, autonomous robotic systems can represent and reason about aspects including the robot's capabilities, their environments, the objects they are to act on, their actions and the effects they cause, and other agent in the environment. Thus, a typical system should be capable of

Envisioning: inferring all possible events and effects that will happen if a plan is executed in a hypothetical situation.

Query answering: given some knowledge preconditions for plan execution inferring pieces of knowledge that satisfy these knowledge preconditions.

[1] Note that we are not concerned with *hardware* reconfiguration here, only the ability of the system to re-route data and control via different software organisation.

Diagnosis: inferring what caused a particular event or effect during execution.

Reconfiguring: re-organising the software architecture within the system, and amending the corresponding expectations and capabilities.

While sub-symbolic components, such as feedback control systems, are vital for coping with varying and dynamic environments, handling all the above within purely sub-symbolic architectures is difficult. For example, in some situations a particular control system (say for a 'gripper') may need to be swapped for another. This behavioural change is very difficult to handle within purely sub-symbolic architectures especially as we wish to retain the scrutability of the architecture. Consequently, many practical autonomous systems have a *hybrid* architecture, where continuous control sub-systems are overseen by an *agent*, that is an high-level decision maker, able to make choices and to provide reasons for them.

The agent makes decisions about actions and plans to undertake, about adaptive controllers and hardware configurations, and about how it can effectively achieve its key goals. This is facilitated by modules that manage the key interactions between the continuous (sub-symbolic) and discrete (symbolic) parts of the system.

This paper provides an overview of a hybrid agent architecture for autonomous systems, where the agent is both self-aware and self-modifying. It also reports on practical autonomous robotic systems based on this architecture.

2 Architecture

Our approach is based on the underlying *modularity* of the system's components.

2.1 Modularity

An autonomous system is made up of a wide range of components, some hardware, some software. These include: sensors, providing information about the environment; actuators, acting upon the environment; control systems, software interfacing with hardware; learning systems; planning systems; etc. In such a system *modularity* is becoming increasingly important, particularly for maintenance and reconfigurability.

The most common framework for practical autonomous systems is the *Robotics Operating System* (ROS) [Quigley *et al.*, 2009]). This provides a framework in which a wide range of software components can be linked together. Importantly, ROS modules link together in the *same* executable in order to pass data between them. The distinct modules are called *nodes*, which communicate through *messages*. A module that wants to receive particular information must subscribe to that specific *topic*. The communication might also happen by individual request, with responses established through *services*.

A key benefit of the modularity provided by a ROS system is that it minimizes the many difficulties in debugging. It also aids incremental development; while implementing a robotics system, we need multiple software components to coexist (i.e., different sensors, different control systems, ..). With a ROS architecture it is possible to run nodes undergoing development alongside pre-existing or well-debugged nodes, and only the node that has been modified needs to be restarted, while ROS itself handles the modifications concerning the whole system infrastructure.

This modularity will also increase the number of possible configurations of the system architecture, greatly improving the flexibility of the system.

2.2 Hybrid Agent Architecture

Based on the inherent modularity provided by middleware such as ROS, we organise the system's architecture so that there are a range continuous control systems over-seen by a single agent (see Figure 1). The agent makes high-level, and essentially discrete, decisions based on the information provided by the feedback control systems (engaged in continuous feedback with the environment). These feedback control systems cover a wide range, from object recognition, sensor fusion and learning through to navigation, manipulation and planning.

Note that the agent makes high-level decisions, such as (for example) *where to go to*, *whether to stop and re-charge*, *what to do in an emergency*, and so on. Crucially, vast amounts of data is *not* passed on the agent. The agent makes its decisions and then invokes further control systems to carry these out. Such hybrid architectures are increasingly popular in the implementation of an autonomous system. Not only is modularity important, but the separation of key decision-making into a distinct (and often verifiable) entity gives clarity over where responsibility lies and what decisions will be made.

An 'agent' is an autonomous computational entity making its own decisions about what action to undertake [Wooldridge, 2002]. Often this means having goals and involving other agents in order to accomplish these goals. However, in many occasions, something more is needed: rather than having a system that just makes its own decisions, a *rational agent* must have explicit reasons (that could also explain, if requested) for choosing one option over another. Since agent are autonomous, understanding *why* an agent chooses an action is very important especially if we want to carry out detailed analysis, even *formal verification* [Fisher *et al.*, 2013].

Such agents make decisions about what action to perform, given their *beliefs*, *desires* and *intentions*. Desires will be derived from the long-term goals, beliefs will depend upon information provided from sensors/control, a *world model* will represent the agent's information about itself, other agents and its environment, while intentions represent the goals that the agent is actively pursuing. This approach has been encapsulated within the BDI model [Rao and Georgeff, 1995; 1991; 1992], where BDI stands for beliefs, desires and intentions. The declarative nature of the BDI approach provides clarity in terms of motivation ("why a certain choice is made") and belief ("what information the agent has").

There are several agent programming languages and agent platforms based on the BDI approach; we use the `Gwendolen` language [Dennis, 2017]. This is essentially a plan selection language where a plan is selected for execution in order to reach a goal (specifically, an 'intention'). After executing a plan, the beliefs and intentions might change, as the agent performs some action in its environment. However, the rational agent, does not, itself, generate plans, but it can in-

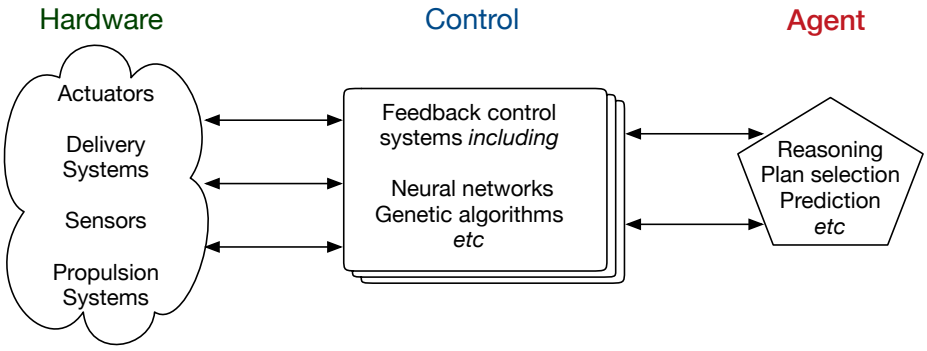

Figure 1: Schematic Overview of Hybrid Agent Architecture.

voke a range of different planning and advisory sub-systems and then makes decisions about which plan is to be executed and when. Therefore, a drawback of this form of approach is that it is essentially a plan management and plan selection framework with no in-built mechanisms for learning. This means that a BDI agent does not automatically learn from past behaviour and adapt its plans accordingly.

2.3 Reconfiguration

The rational agent is also *self-aware*. It has a *world model* that includes a description of the autonomous systems' architecture. Once it has invoked particular control systems it typically just monitors activity within the sub-systems. It will be only if something crucial changes, or fails, that the agent will need to perform additional reasoning, for example deciding to replace a plan (or part of it). In the case of significant failure/change, more drastic modification may be required, such as changing the ROS streams and nodes. Before changing such a component, the agent needs to be aware of what the current component offers, what the new component offers instead, what the new component requires, and how it interacts with other components.

Thus, the rational agent has to be able to:

- reason about beliefs;
- select plans based on beliefs;
- react to change in the world model (i.e. loss or addition of capabilities);
- monitor the system's performance; and
- configure the system in order to be able to achieve mission goals and overcome deficiencies.

The reconfiguration is crucial — such autonomous systems have to be self-aware, but also able to modify the hardware or the software configuration, especially if the system will be used in hazardous or distant environments, where the human intervention is not possible. The reconfiguration process is required when the system needs to cope with changes, in the hardware or in the environment, or to adapt to failures or damages of some of the sub-systems, potentially allowing the agent to still achieve its goal. The modular nature of the agent-based architecture makes the reconfiguration process more feasible. Considering the architecture of the system we have identified different reconfiguration possibilities [Dennis *et al.*, 2014]:

- Reconfiguration *due to the hardware*: if some hardware fails, or is possibly even added, the agent needs to modify the control systems or its high-level decisions, in order to take in account respectively the restricted or increased possibilities.
- Reconfiguration *due to the control*: this kind of reconfiguration may occur if the agent detects errors in software controllers while the system is running, or finds new controllers with better performance. In this case, the agent can again reconfigure its high-level goals/plans upon the new control systems. ROS provides a perfect framework for reconfiguration, by using a modular structure where nodes can be incorporated or removed easily.
- Reconfiguration *due to the agent*: in this case the hardware and control systems remain the same, but the agent might reconfigure its high-level elements, such as its goals, plans, word model (knowledge base), or changing some module such as its preferred planner or the learning module to be used.

3 Implementation

This hybrid agent architecture has been instantiated in a number of application areas, and for a number of different purposes.

Space. In [Lincoln *et al.*, 2013] we described how autonomous satellites might be implemented including this approach. This was particularly useful in supporting formal verification [Dennis *et al.*, 2016] and in showing that this approach aids readability and conciseness [Dennis *et al.*, 2010a]. The architecture was used in both simulations and real hardware.

Automotive. In [Kamali *et al.*, 2017] we showed how this architecture could also be used for autonomous road vehicles, particularly *convoys of driver-less vehicles*. Again, verifiability was a key target and, again, the architecture was used in both simulations and real hardware.

Aerospace. While the focus of our development of unmanned air vehicles was again formal verification, this

Figure 2: Test Rig at National Nuclear Laboratories, UK.

was with a specific purpose. We used the verifiability of the architecture to provide evidence for the certification of unmanned air vehicles [Webster *et al.*, 2014]. In this case, only software simulations were developed.

Nuclear. In [Aitken *et al.*, 2017] we instantiated the architecture for use in *autonomous nuclear decommissioning*. This time we were not concerned with verification but particularly targeted *reconfigurability* [Dennis *et al.*, 2014] since being able to continue working without human intervention is crucial if deployed in a nuclear environment. This was implemented both in simulation and in hardware, and is currently being deployed in a nuclear test environment in the UK — see Figure 2.

It is this last application, where the flexibility and resilience provided by self-reconfigurability is crucial, that we will concentrate on as an exemplar.

3.1 Agent with abstract engine

A key problem when connecting continuous control systems, such as our range of sub-symbolic modules, to a discrete entity, such as our agent, is that a continuous stream of data (for example, coming from sensors) must be handled and converted into discrete values that the agent can reason about. This has led to a practical modification of the basic hybrid agent architecture [Dennis *et al.*, 2010b], as outlined in Figure 3.

The key addition is an *abstraction engine*, sitting between the *reasoning engine* (aka the *agent*) and the rest of the system. This abstraction engine provides the *continuous* to *discrete* translation, taking streams of data from the sub-symbolic subsystems and passing on discrete abstractions of this to the agent itself. Specifically, it aims to identify data of interest to the agent and package this together into a concise form.

As the agent deals only with discrete information, the abstraction engine has to abstract the data from the control subsystems (quantitative values such as distances, or coordinates) into predicates (i.e., *toofar*, *tooclose*, etc). The abstraction engine is also responsible of translating all the decisions and action invocations coming from the agent into proper commands for the control subsystems.

In several applications, both the abstraction engine and the agent are implemented in a variant of the `Java`-based `Gwendolen` agent programming language [Dennis, 2017], that support BDI-based programming. The communication between the `Java` process and the control subsystems is via ROS messages, and exists within a `Java` "environment" layer [Dennis, 2014].

3.2 Knowledge base

The autonomous system contains not only an agent, but also a knowledge base (or world model), which stores all the information that need to be shared by the various high-level systems and control sub-systems to allow their collaboration. The knowledge base is modular with several different sections. These include a section for *perception*, consisting of ground literals describing information about the external world, and coming from sensors and diagnostics. In addition, the knowledge base also stores a description of the existing components in the system and their capabilities (i.e. a *configuration*). Lastly, a module, described as *program data*, stores all the models, route plans, maps, metrics, and other miscellaneous data that must be shared by the components within the autonomous system.

Essentially, the agent receives data from the knowledge base and, based on its internal decisions, sends commands through the abstraction engine which interfaces the robot's control systems. The connection between the rational agent and the knowledge base is managed via two mechanisms.

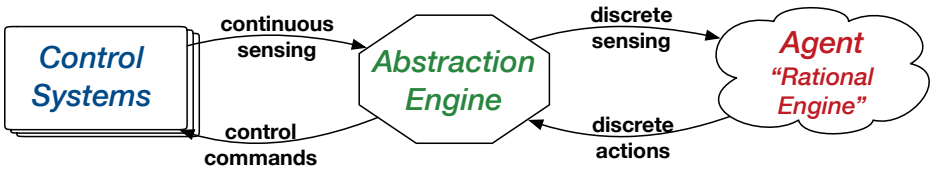

Figure 3: Architecture of an hybrid agent system.

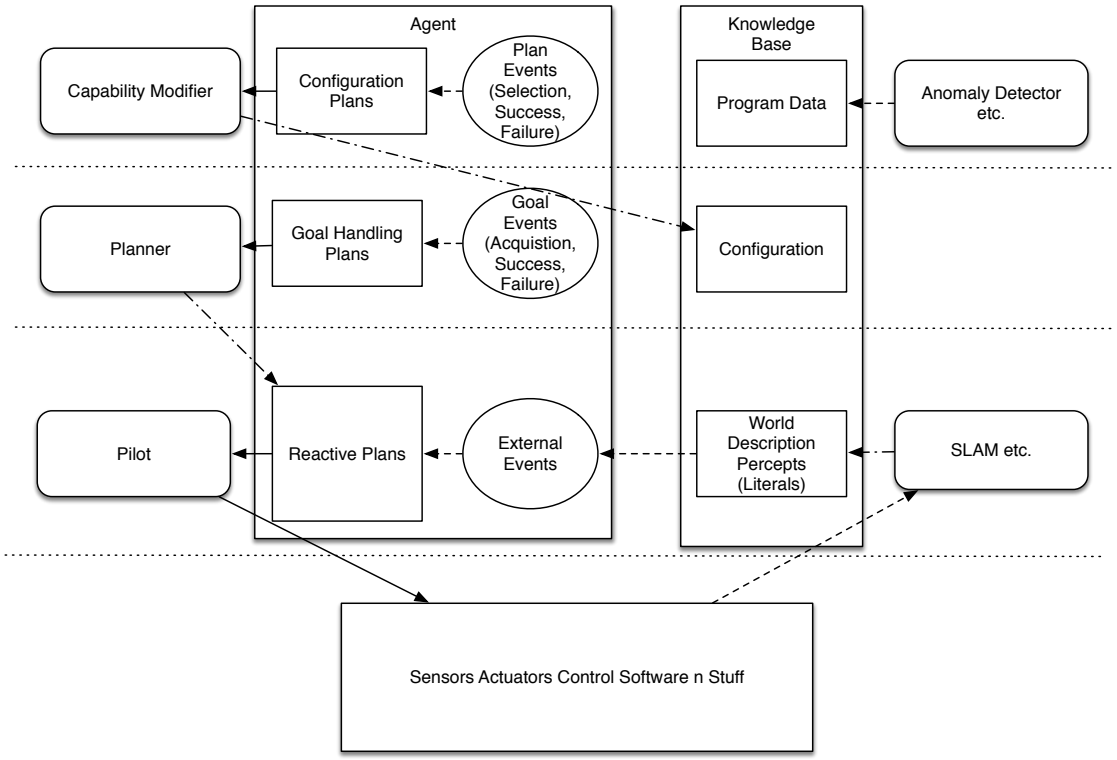

Figure 4: A layered view of the system.

The agent can register an interest in particular ground facts and will be notified on a "push" basis whenever those facts change. Furthermore, it can query the knowledge base for additional information. In this way the agent is not over-loaded with information about every change that occurs in the knowledge base, but it can still access any information it needs and learn every information of critical importance as soon as possible.

The description of the whole autonomous system, as captured in the knowledge base, is hierarchical. This description can be viewed as a sequence of layers (as the traditional three-layer architecture common in robotics), as it is shown in Figure 4. The 'lowest' layer sits on top of the control systems and consists of reactive plans, which monitor external events (based on changes in the perception module of the knowledge base) and send commands to components which control actuators. The 'middle' layer consists of planners and simulators. These components use information about the configuration (which are not used by the reactive plans) of the system and the output from these components become new reactive plans. Plans in this layer are activated by monitoring and analysing the agents' goal.

Finally, at the 'top' layer, we have tools that react to problems with the configuration. This layer monitors the success or failure of specific plans in the layers below and, if necessary, can reconfigure the underlying system. In the particular case of capability modification, the reconfiguration simply changes the description that is stored in the knowledge base, but in order to obtain this modified description it may need to acquire new information about the performance of the system using the anomaly detector.

3.3 Reconfigurability example

Consider the following example. The agent believes it recognises a certain object that it wishes to pick up and place in a container. This recognition is provided by sensor and object recognition sub-systems. The agent makes the decision to invoke a robot arm to grasp the object and move it to a specified container. So, it sends the appropriate command, via the abstraction engine, to the arm control sub-system. However the grasping and movement operation fails.

Now the agent must assess, as best it can, *why* this failed. Was it

- because of a problem with the robot arm (e.g. it did not move to the correct position)?
- because of a problem with the gripper (e.g. it failed to grasp the object even when at the correct position)?
- because of a problem with the object recognition software (e.g. it mis-categorised the object)?
- because of a problem with the cameras (e.g. they misrecorded the position of the object)?
- *etc.*

This is, of course, quite difficult but, let us assume that after a series of failures the agent reasons that the most likely culprit is the vision system (for instance, after checking proper values the agent becomes aware that the camera has noise). And, let us assume that the system has two other cameras available. These are not as precise as the one in use, but when used together can give reasonably accurate (though not as precise) visual information. Thus, the agent can

1. reconfigure the software architecture, so that the two alternative cameras are switched on and visual data is taken from those two rather than the original one,
2. change the description of the visual component not only to reflect the new cameras but also to reflect the reduced accuracy/capability, and
3. revisit the plans and goals it has to ensure that this reduced capability is sufficient to achieve what is required.

If there are problems with this last item, then the agent might invoke one or more planning sub-systems to re-plan the achievement of goals with the new reduced capability. And so on.

Typically, once the agent has carried out some reconfiguration of this form it will monitor decisions, information and outcomes to see if this change has, indeed, improved system accuracy. If it turns out that it has actually reduced accuracy then the agent may choose to change the architecture back again.

3.4 Autonomous industrial robotic arm

We used the architecture described to implement an agent-based system able to assist an autonomous nuclear waste management process [Aitken *et al.*, 2017]. This application scenario is, in particular, dangerous for direct human intervention, and is also a very mundane and repetitive task. Therefore, an autonomous robot, able to carry out the process, will be very important. This system is developed in order to perform a "sort and disrupt" task and integrates a vision system, a robot arm and a rational agent, all connected using the ROS framework. The system should sort through nuclear waste items, placing them in containers as appropriate. In some cases the system may need to *disrupt* an item, e.g. break open a large metal container to find what is inside.

The overall architecture, is as we have described already. The abstraction engine receives information from vision/recognition system (through ROS topics) and then the reasoning engine (aka *agent*) takes decisions about what action has to be done. These decisions are then published to a different topic by the abstraction engine, in order to be available as a command understandable to the robot control subsystems. The abstraction engine also receives information about the performance of the robot. Therefore, knowledge-based decisions need to be made using available data, such as physical attributes of the waste items or vision data collected.

As well as handling the main task, the agent can also reason about faulty equipment or better control systems (that might be available after the deployment of the system over long periods of time). The reasoning engine can then use a reconfiguration strategy to instruct the arm to switch to a different equipment (if the action has been evaluated not successful) or a more updated control system.

This system has been developed in university labs and is now being transferred to the National Nuclear Laboratories site in Workington, UK, for further testing and development; see Figure 5.

3.5 Responsibility and Verifiability

Though not yet used in the nuclear application, a key benefit of the hybrid agent architecture is that there is an identifiable entity which can provide explicit reasons for making decisions (i.e. the *agent*). This means that the decisions that agent might make, such as which containers to put certain items in, can be analysed in detail to see if it matches expected behaviour. In this sense the 'responsibility' for decisions is clear.

Similarly, as most industrial robotic systems must go through complex regulatory and certification processes, then the regulators will be happy to see that not only is there an identified decision-maker but that the reasons for making those decisions are explicitly and open. This will likely significantly aid *certification* and the need for complex autonomous systems to be *scrutable* in this way [Caminada *et al.*, 2014] will surely be essential in the future.

In general, this approach, with an identified rational agent, provides a much clearer route to verification. Indeed, we have developed and carried out formal verification for a range of architectures of this form [Fisher *et al.*, 2013; Dennis *et al.*, 2016; Kamali *et al.*, 2017]. This is important not only for certification issues, as above, but for general reliability, safety, etc.

4 Conclusions

In this paper we described an hybrid architecture for autonomous systems in which control systems are over-seen by an high-level decision maker, a rational agent. This modular

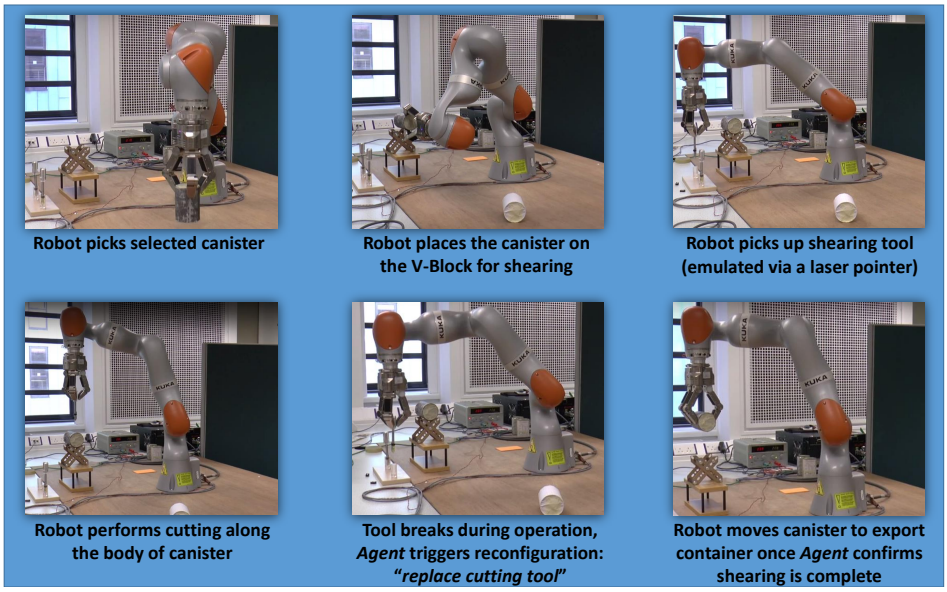

Figure 5: Video Capture of Demonstrator Process.

architecture not only ensures that key decision-making (and so, *responsibility*) is isolated in a single entity, but also allows the reconfiguration of mission goals, capabilities, and control sub-systems at run-time. The ability of an autonomous system to reconfigure its own architecture is crucial if the system is to be deployed in areas where the direct human intervention and repair is not feasible.

As well as providing an overview of our developments in this area, we also explored how the different parts of the system communicate between themselves. This architecture has been used in several systems and we specifically highlight the implementation of an autonomous robotic arm used for complex tasks in extreme or hazardous environments. The nature of the activity demands that an autonomous system, able to make decisions and to self-reconfigure, is deployed, as it must operate for long period of time, performing operations that are both repetitive and dangerous for human operators.

References

[Aitken *et al.*, 2017] Jonathan M. Aitken, Affan Shaukat, Elisa Cucco, Louise A. Dennis, Sandor M. Veres, Yang Gao, Michael Fisher, Jeffrey A. Kuo, Thomas Robinson, and Paul E. Mort. Autonomous Nuclear Waste Management. (Under review), 2017.

[Caminada *et al.*, 2014] Martin W. A. Caminada, Roman Kutlák, Nir Oren, and Wamberto Weber Vasconcelos. Scrutable Plan Enactment via Argumentation and Natural Language Generation. In *Int. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1625–1626. IFAAMAS/ACM, 2014.

[Dennis *et al.*, 2010a] L. A. Dennis, M. Fisher, N. Lincoln, A. Lisitsa, and S. M. Veres. Reducing Code Complexity in Hybrid Control Systems. In *Proc. 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-Sairas)*, 2010.

[Dennis *et al.*, 2010b] Louise A. Dennis, Michael Fisher, Nicholas Lincoln, Alexei Lisitsa, and Sandor M. Veres. Declarative Abstractions for Agent Based Hybrid Control Systems. In *Proc. 8th International Workshop on Declarative Agent Languages and Technologies (DALT)*, pages 96–111, 2010.

[Dennis *et al.*, 2014] Louise A. Dennis, Michael Fisher, Jonathan M. Aitken, Sandor M. Veres, Yang Gao, Affan Shaukat, and Guy Burroughes. Reconfigurable Autonomy. *KI — Künstliche Intelligenz*, 28(3):199–207, 2014.

[Dennis *et al.*, 2016] Louise A. Dennis, Michael Fisher, Nicholas K. Lincoln, Alexei Lisitsa, and Sandor M. Veres. Practical Verification of Decision-Making in Agent-Based Autonomous Systems. *Automated Software Engineering*, 23(3):305–359, 2016.

[Dennis, 2014] Louise A. Dennis. ROS-AIL Integration. Technical Report ULCS-14-004, University of Liverpool, Department of Computer Science, 2014.

[Dennis, 2017] Louise A. Dennis. Gwendolen Semantics: 2017. Technical Report ULCS-17-001, University of Liverpool, Department of Computer Science, 2017.

[Fisher *et al.*, 2013] Michael Fisher, Louise A. Dennis, and Matthew Webster. Verifying Autonomous Systems. *ACM Communications*, 56(9):84–93, 2013.

[Kamali *et al.*, 2017] Maryam Kamali, Louise A. Dennis, Owen McAree, Michael Fisher, and Sandor M. Veres. Formal verification of autonomous vehicle platooning. *Science of Computer Programming*, 2017.

[Lincoln *et al.*, 2013] N. Lincoln, S. Veres, L. Dennis, M. Fisher, and A. Lisitsa. Autonomous Asteroid Explo-

ration by Rational Agents. *IEEE Computational Intelligence Magazine*, 8(4):25–38, 2013.

[Quigley *et al.*, 2009] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an Open-source Robot Operating System. In *Proc. ICRA Workshop on Open Source Software*, 2009.

[Rao and Georgeff, 1991] A. S. Rao and M. P. Georgeff. Modeling Agents within a BDI-Architecture. In *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 473–484. Morgan Kaufmann, 1991.

[Rao and Georgeff, 1992] Anand S. Rao and Michael P. Georgeff. An Abstract Architecture for Rational Agents. In *Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 439–449, Cambridge, MA, USA, 1992. Morgan Kaufmann.

[Rao and Georgeff, 1995] Anand S. Rao and Michael P. Georgeff. BDI Agents: from Theory to Practice. In *Proc. 1st International Conference on Multi-Agent Systems (ICMAS)*, pages 312–319, San Francisco, USA, 1995.

[Webster *et al.*, 2014] M. Webster, N. Cameron, M. Fisher, and M. Jump. Generating Certification Evidence for Autonomous Unmanned Aircraft Using Model Checking and Simulation. *Journal of Aerospace Information Systems*, 11(5):258–279, May 2014.

[Wooldridge, 2002] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.