Editor: **Terry R. Payne,** University of Liverpool, t.r.payne@liverpool.ac.uk

Satellite Control Using Rational Agent Programming

Louise Dennis, Michael Fisher, and Alexei Lisitsa, *University of Liverpool*
Nicholas Lincoln and Sandor Veres, *University of Southampton*

Traditionally a satellite is a large and expensive piece of equipment, tightly controlled by a ground team with little scope for autonomy. The space industry has recently sought to abandon large monolithic platforms, however, in favor of multiple, smaller satellites working in teams to accomplish the task of a larger vehicle through distributed methods. Both financially and functionally motivated, such developments help reduce launch vehicle constraints and nearly eliminate ground station personnel costs, while introducing fault tolerance and redundancy into the system. Moreover, in some instances, a distributed platform is the only feasible method to accomplish a particular mission.

Such distributed space missions are not restricted to the realms of academia. The Cluster mission, launched by European Space Agency (ESA) in July 2000 (see Figure 1), consists of four satellites working collaboratively to investigate the interaction of the Earth's magnetosphere with the solar wind. The Cluster mission represents a highly suitable application of distributed hardware in an Earth orbit, namely that of a sensor web. Additional interest in Earth orbiting sensor webs is mainly focused on Earth observation, which NASA's EO-1 program is currently investigating. More adventurous multisatellite implementations involve exploration scenarios, such as the exploration and cataloguing of the asteroid belt between Mars and Jupiter. ESA's proposed Apies mission seeks to achieve this goal via a cluster of spacecraft moving through the dynamic asteroid field.[1] Such a mission clearly requires a high degree of autonomy, not only to enable successful navigation of the asteroid environment, but to dynamically assign specific spacecraft assets to asteroids of particular scientific interest.

With the implementation of collaborative space vehicles, design complexity has moved from being restricted to the space vehicle's physical build to being primarily within satellite operations. That is, issues such as interdistance regulation and dynamic navigation are tasks we cannot manage from a control room due to timeliness constraints. System autonomy, or at the very least semiautonomous action, is required wherein operators provide a satellite with high-level instructions for a specific task, which it then carries out autonomously. Introducing such (semi)autonomous behavior into a multiple satellite systems presents challenges to the development of appropriate control software and, in particular, the reliability of such systems.

An ongoing study at the Universities of Liverpool and Southampton is looking at using agent programming technology to control complex autonomous satellites. In this work, we are exploring how well rational agents cope with autonomous decision making in continuous systems. The rational agent aspect ensures that autonomy, its control, and its requirements are all clear and explicit.

Control Systems Technology

A fundamental component of control systems technology is the feedback controller. This measures, or estimates, a system's current state through a dynamic model and produces subsequent feedback and feed-forward control signals. In many cases, difference and differential equations can be used to elegantly manage the process. These equations of complex dynamics make changes to the input values of subsystems and monitor the outcomes on various sensors.

Such controllers are increasingly required to work in situations where there are areas of discontinuity, when the situation requires a distinct

1541-1672/10/$26.00 © 2010 IEEE
Published by the IEEE Computer Society

change in behavior and often needs control to switch to the use of an alternative model and alternative control equations. This kind of hybrid control system clearly requires that we integrate some decision-making system with the feedback controller.[2–4] It might also be necessary for a system to take actions, such as detecting that a fuel line has ruptured and switching valves to bring an alternative online, that fall outside the scope of monitoring and adjusting input and output values but involve detecting that thresholds have been exceeded or making large system changes. It is by now well established that using a separate discrete and logical decision-making process for this aspect is preferable to greatly extending the basic control system.[5,6] Overall systems with these characteristics are often referred to as *hybrid control systems* in that they integrate discrete, logical decision processes with physical system dynamics.

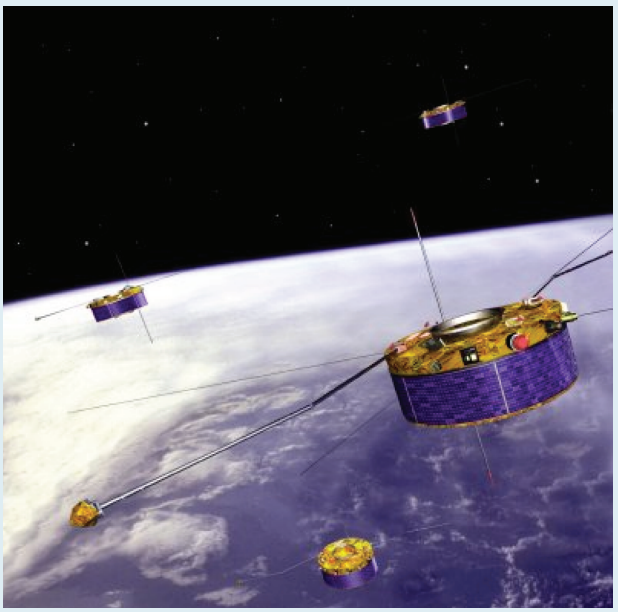

Figure 1. Artist's illustration of the European Space Agency (ESA) Cluster mission. Multiple, smaller satellites working in (semi) autonomous teams can help space exploration become more financially and functionally efficient. (Courtesy of ESA image archive)

Unfortunately, controlling hybrid systems with traditional programming methods can become increasingly unwieldy. Researchers often represent the decision process as an inflexible tree (or graph) of possible situations. Execution then involves tracing through a branch that matches the current situation and then executing the feedback controller (or making other system changes) found at the relevant leaf of the tree. Figure 2 illustrates such a hybrid automaton in StateFlow.

Programming these decisions from state to state is often time-consuming and error-prone and can lead to the duplication of code where the same actions must be taken in several, slightly different situations. There are at least two ways to depart from this basic programming decision paradigm within hybrid systems. One approach arises out of the field of subsymbolic artificial intelligence, where neural nets or genetic algorithms can be used to either automatically generate or compactly control such systems.[7,8] However, such techniques obscure the decision-making process, so it is no longer easy to tell why the system is operating the way it is. This causes obvious problems for debugging, diagnosis, monitoring, and most importantly, reliability, which is a fundamental requirement in expensive space missions. Figure 3 illustrates this approach and highlights that techniques imitating natural intelligence are used to direct these hybrid control systems.

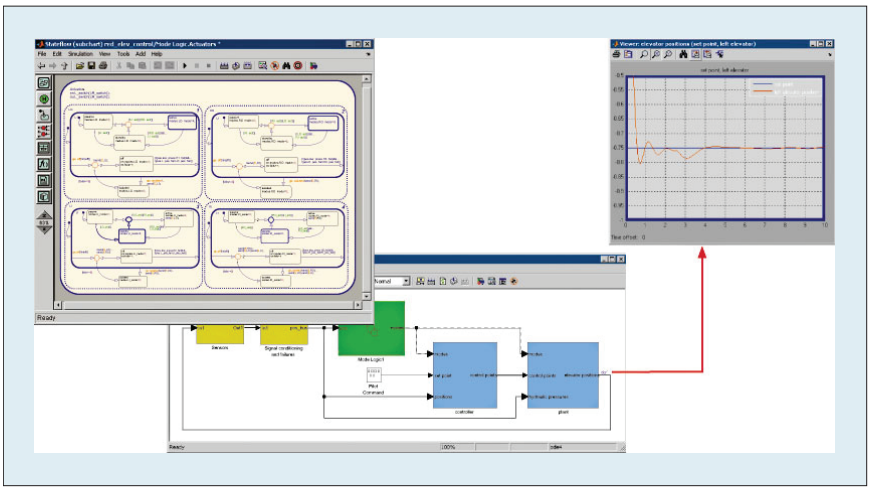

Figure 2. Diagrammatic hybrid system description in StateFlow. This software is one of a few software systems available that permit graphical editing and simulation of hybrid systems. Guard conditions are used to define when the system is to transition into another state within a finite-state machine. The hybrid system's actual state includes both continuous and discrete variables. (Courtesy of Mathworks)

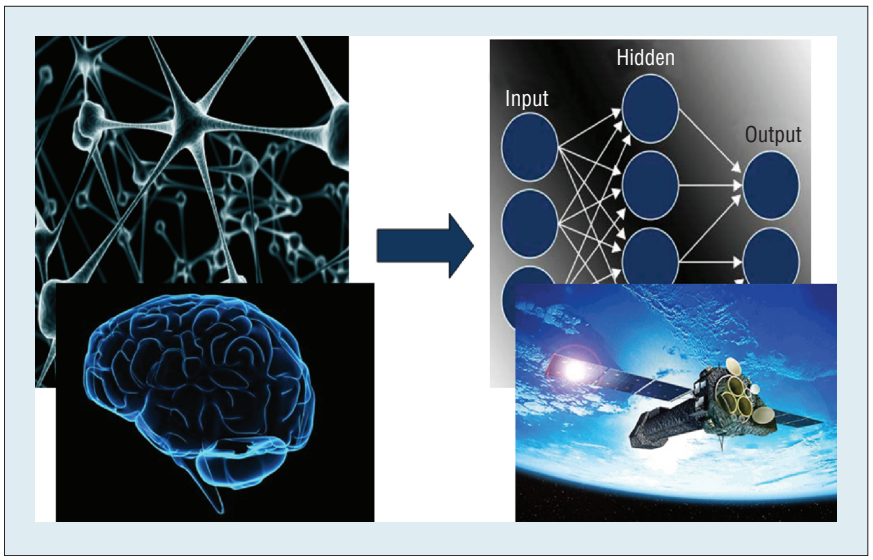

Figure 3. Neuro/fuzzy solutions for decision and control. These methods can provide solutions to nonlinear feedback control problems and also help create abstractions from the continuously sensed world to discrete logic statements. By their very nature, they are less suitable for declaring goals and maintaining behavior rules for autonomous systems.

An alternative approach to simplifying autonomous decision making in a hybrid system involves carefully choosing abstractions that relate the continuous world with discrete decision states. Using these abstractions, we can define basic rules of behavior and then formulate goals to keep the system within constraints and set both short- and long-term objectives. This lets us use an agent-based approach, where goals, plans, and logical inference are all captured within a rational agent (see the "Agent Programming" sidebar). Using such an approach requires a carefully constructed set of hierarchical abstractions (see Figure 4). It not only provides clear and coherent decision making, it also emphasizes the process's autonomous nature. Thus, using an agent-based programming approach, the choices the agent makes are visible and explicit, as are the reasons it has for taking them.

Rational Hybrid Agents

Our aim is to produce a hybrid system embedding existing technology for generating feedback controllers and configuring satellite systems within a decision-making part based on a high-level agent programming language. Such languages assume an underlying imperative programming layer in which an agent's actions are executed. Hybrid control systems appear to be a natural fit for this programming style in which a decision-making layer is combined with a lower-level, dynamic execution layer.

Decision making tends to rely on discrete information (such as "a thruster is broken"), while system control tends to rely on continuous information (such as "thruster fuel pressure is 65.3"). Thus, it is vital to be able to abstract from the dynamic system properties and provide discrete abstractions for use by the agent program (see Figure 4). For this reason, our architecture has an explicit abstraction layer that translates between the two information styles as data flows around the system. The *abstraction engine* generates a stream of incoming sensor and action abstractions, using the sEnglish

Agent Programming

Systems that combine aspects of autonomy, concurrency, and communication are notoriously difficult to program. Early attempts using conventional programming techniques produced systems that were complex, unclear, and frequently error prone. The agent paradigm grew out of an attempt to find appropriate abstractions for describing and structuring these systems. Agent programming separates processes into separate entities each with its own supply of facts (or beliefs) about the larger system and its own procedures for executing its role in the system. Rational agent systems are a major strand within agent programming that seek to put programming within a framework of goals, deliberation, and explainable decision making. Consequently, high-level languages have been developed to support this programming style by providing constructs for goals, beliefs, and plans.[1,2] Many of these languages extend from work in declarative programming, so a rational agent program can often be viewed as a logical specification of the desired behavior, thus opening up the possibility for formal verification of the resulting code.[3]

References

1. R.H. Bordini et al., eds., *Multi-Agent Programming: Languages, Platforms and Applications*, Springer, 2005.
2. R.H. Bordini et al., eds., *Multi-Agent Programming: Languages, Tools and Applications*, Springer, 2009.
3. R.H. Bordini et al., "Model Checking Rational Agents," *IEEE Intelligent Systems*, vol. 19, no. 5, 2004, pp. 46–52.

ontology language (see the related sidebar), which control engineers already use.

In our system's architecture, a traditional feedback controller governs the real-time satellite control. This forms a *physical engine*, which sends data to an *abstraction engine* that filters and discretizes information from both the environment and the physical engine. To achieve this, the abstraction engine might also call on a *continuous engine* to make calculations involving the continuous data. Finally, a *rational engine* uses rational agent technology to make decisions about both the system configuration and its parameters that are transmitted to the physical engine. The rational engine can call the continuous engine (via the abstraction engine) to, for instance, generate new controllers or can send instructions directly to the physical engine.

The agent programming language within the rational engine encourages an engineer to express decisions in terms of the facts an agent has on hand, what it wants to achieve, and how it will cope with any unusual events. This reduces code size so engineers need not explicitly describe how the satellite should behave in each possible system configuration and can instead focus on describing the decisions relevant to particular configurations. The key aspect of deliberation within agent programs lets the decision making part of the hybrid system adapt intelligently to changing dynamic situations, priorities, and uncertain sensors.

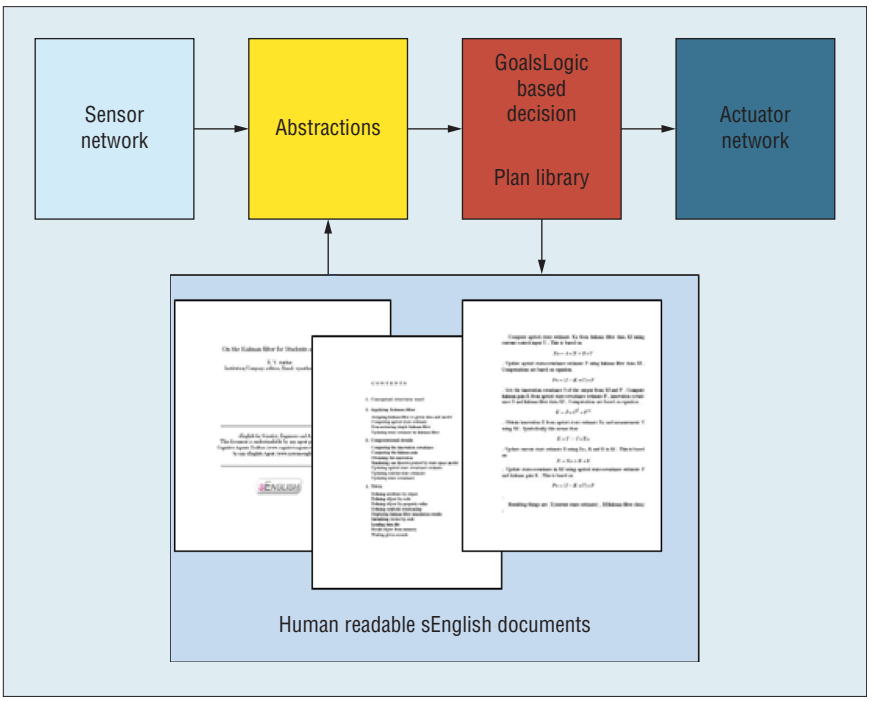

Figure 4. Abstractions processes in the proposed agent-based approach to hybrid control systems. Such an agent-based programming approach makes the agent's choices and reasoning both visible and explicit.

Case Study: Satellite in Geostationary Orbit

Our first case study involved a single satellite attempting to acquire and maintain a geostationary orbit. A geostationary orbit, commonly used for communications satellites, requires active maintenance because solar radiation and disturbing gravitational forces act on the satellite. We could achieve such maintenance using a

sEnglish

System English (sEnglish) is a controlled natural language—that is, a subset of English with meanings of sentences defined by code in a high-level programming language such as Matlab, GNU Octave, SciLab, Python, or C++.[1,2] sEnglish is also an example of natural language programming; correctly formulated sEnglish text compiles into executable program code unambiguously so long as predefined sentence structures and an ontology are defined, and errors in functionality are reduced due to the structures inherent within sEnglish. This lets a programmer enjoy the convenience of natural language while retaining the usual determinism of digital programs. Once a database of sentences and ontologies have been generated, the clarity and configurability of a system written in sEnglish becomes evident. Of particular interest, when applied to agent system development, is the link between the abstract manner in which sEnglish solutions are developed and the abstractions of an agent system. This enables a shared understanding to be provided between the satellite and its operator.

References

1. S.M. Veres, *Natural Language Programming of Agents and Robotic Devices: Publishing for Humans and Machines in sEnglish*, SysBrain, 2008.
2. L. Molnar and S.M. Veres, "Documents for Intelligent Agents in English," *Proc. IASTED Conf. Artificial Intelligence and Applications* (AIA 2010), Int'l Assoc. Science and Technology for Development, 2010, pp. 674–122.

traditional feedback controller, but a thruster failure might (due to fuel venting) rapidly move the satellite out of its prescribed orbital location. Diagnostic reasoning about such occurrences and subsequent reconfiguration of the control hardware is necessary to compensate for such events and allow for mission continuation and completion.

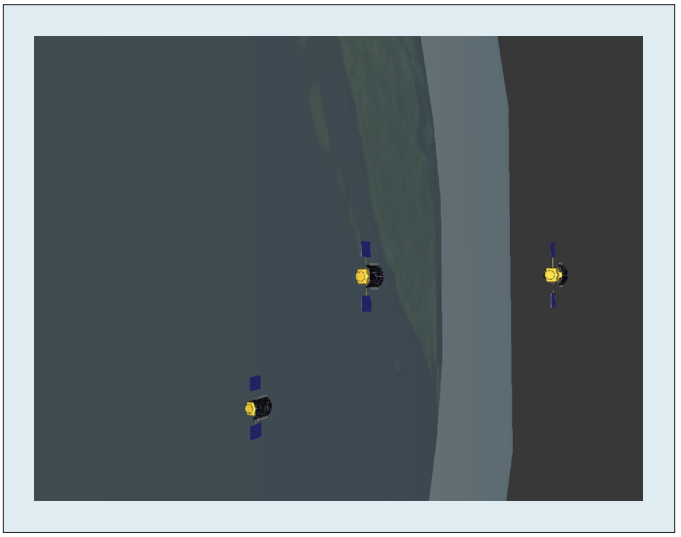

Figure 5. Virtual reality Matlab Simulink display of three controlled satellites in a low Earth orbit. Each satellite is controlled by an agent architecture that must diagnose and react to simulated thruster failures.

In implementation, we programmed the continuous engine with routines for calculating bound intersections and producing a minimum fuel path that would bring the satellite back to the desired operational orbit. In the event of a thruster failure, the reasoning engine deduces a suitable new hardware configuration and communicates this to the physical engine. Concurrent with these system diagnostics, the reasoning engine works with the continuous engine to determine if the satellite has strayed "out of bounds." Based on this evaluation, the reasoning engine triggers the production and execution of a new fuel optimal path or switches to a position regulatory feedback controller. In all instances, the physical engine executes control procedures after receiving direct instructions from the reasoning engine.

The rational agent language we used in the reasoning engine let us handle these situations (hardware reconfiguration and control implementation) separately in the code, thus reducing the programming complexity. The agent technology also permitted the same code to be used to reason about any of the thrusters, rather than duplicating the code for each thruster, something that many current tools for handling hybrid systems do not allow.

The simulation injected the satellite into the orbital position with a bounded error and tasked it with acquiring and maintaining a constant position relative to the Earth (within certain bounds) at a geostationary altitude. During the operational mode, the satellite was also subject to potential failures in its thrusters, ranging from short circuits and gain reduction to more dramatic scenarios of burst fuel lines. Upon activation of the agent system, the simulated satellite was able to recover the desired orbital location and regulate this position, while concurrently dealing with thruster failure modes.

Case Study: Multiple Satellites

The current architecture is being extended to multiple satellites that communicate information among themselves while operating in a dynamic environment. We are interested in both the techniques required to maintain a formation and how groups of satellites can implement fault tolerance in the satellite system by switching team roles and altering formations to compensate for equipment failure. This will let us investigate the application of agent techniques for cooperation, coordination, and teamwork.

This multiple satellite scenario involves three low Earth orbiting satellites, as Figure 5 shows. Each satellite is controlled by the previously described agent architecture and may experience thruster failure modes as in the geostationary scenario. We tasked the satellites with completing a scientific mission; the reasoning engine was responsible for diagnostic tasks and coordinating satellite activities to achieve the prescribed scientific mission.

The software is to be evaluated on a physical satellite simulation environment developed at the University of Southampton (see Figure 6). Although this environment constrains the satellites to operate with five degrees of freedom, it lets us assess the software in a real physical environment and evaluate its decision-making ability outside an entirely virtual environment.

As this research progresses, we aim to tackle increasingly more complex and realistic autonomous space software scenarios. We are also developing a high-level agent programming language deployable in the Cognitive Agent Toolbox for Matlab (see www.sysbrain.com) and customized for implementing autonomous control in hybrid systems. Abstraction is clearly important, and we aim to

provide a principled database query solutions for flagging the most relevant abstractions for the rational engine.

Based on our previous work in agent verification,[9] we aim to address the formal verification of both the reasoning engine and various forward-planning techniques. These would potentially let agents reason about the outcome of possible actions and use this information in their decision-making process. If successful, this work could theoretically be extended to agents with the capacity to learn both new abstractions and new plans.

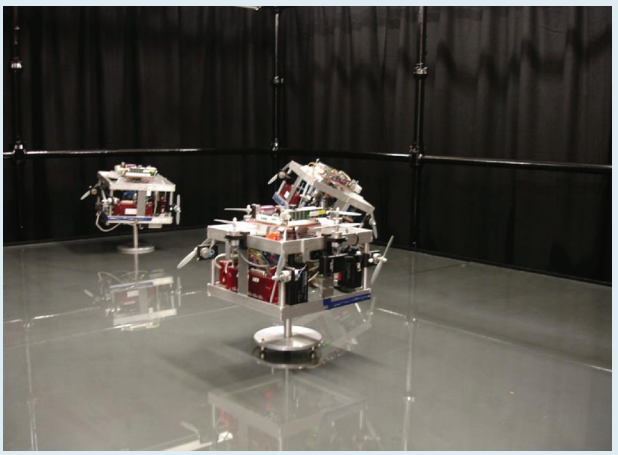

Figure 6. The Autonomous Systems Facility based at the University of Southampton. This real physical environment lets the developers test the software and assess its decision-making ability outside a virtual environment.

Acknowledgments

This research is part of the Engineering Autonomous Space Software project, which is supported within the UK by the Engineering and Physical Sciences Research Council (EPSRC) under grants EP/F037201 and EP/F037570 and supported in part by the European Space Technology Center of the European Space Agency.

References

1. P. D'Arrigo and S. Santandrea, "The APIES Mission to Explore the Asteroid Belt," *Advances in Space Research*, vol. 38, no. 9, 2006, pp. 2060–2067.
2. M.S. Branicky, V.S. Borkar, and S. Mitter. "A Unified Framework for Hybrid Control: Model and Optimal Control Theory," *IEEE Trans. Automatic Control*, vol. 43, no. 1, 1998, pp. 31–45.
3. R. Goebel, R. Sanfelice, and A. Teel, "Hybrid Dynamical Systems," *IEEE Control Systems Magazine*, vol. 29, no. 2, 2009, pp. 28–93.
4. P. Varaiya, "Design, Simulation, and Implementation of Hybrid Systems," *Proc. 20th Int'l Conf. Application and Theory of Petri Nets*, Springer, 1999, pp. 1–5.
5. R. Alur et al., "The Algorithmic Analysis of Hybrid Systems," *Theoretical Computer Science*, vol. 138, no. 1, 1995, pp. 3–34.
6. R. Alur et al., "Discrete Abstractions of Hybrid Systems," *Proc. IEEE*, vol. 88, IEEE Press, 2000, pp. 971–984.
7. P.J. Fleming and R.C. Purshouse, "Genetic Algorithms in Control Systems Engineering," *Proc. 12th IFAC World Congress*, Elsevier Science, 2001, pp. 383–390.
8. F.W. Lewis, S. Jagannathan, and A. Yesildirak, *Neural Network Control of Robots and Non-linear Systems*, Taylor and Francis, 1999.
9. R.H. Bordini et al., "Automated Verification of Multi-Agent Programs," *Proc. 23rd IEEE/ACM Int'l Conf. Automated Software Engineering* (ASE), IEEE Press, 2008, pp. 69–78.

Louise Dennis is a researcher at the University of Liverpool. Contact her at L.A.Dennis@liverpool.ac.uk.

Michael Fisher is a professor at the University of Liverpool. Contact him at MFisher@liverpool.ac.uk.

Alexei Lisitsa is a lecturer at the University of Liverpool. Contact him at lisitsa@liverpool.ac.uk.

Nicholas Lincoln is a researcher at the University of Southampton. Contact him at n.k.lincoln@soton.ac.uk.

Sandor Veres is a professor at the University of Southampton. Contact him at s.m.veres@soton.ac.uk.