Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

A Soar-based Planning Agent for Gas-Turbine Engine Control and Health Management

Paolo Gunetti*. Haydn Thompson**

**Department of Automatic Control and Systems Engineering, University of Sheffield; Mappin Street, S1 3JD, Sheffield, UK (Tel:0044-(0)1142225633; e-mail: p.gunetti@sheffield.ac.uk).*
*** Department of Automatic Control and Systems Engineering, University of Sheffield; Mappin Street, S1 3JD, Sheffield, UK (Tel:0044-(0)1142225649; e-mail: h.thompson@sheffield.ac.uk)*

Abstract: Intelligent Agent technologies constitute an important stream of research in the Artificial Intelligence community. Their characteristics make them suitable for a variety of applications. In this paper, we investigate the use of Intelligent Agent technology in the field of Gas-Turbine Engine Control and Health Management. We present and test a Planning agent, developed to choose and apply appropriate investigative and reversionary action plans, which are useful to correctly assess and mitigate faults. The agent is based on Soar technology and tests are performed using a previously developed Intelligent Agent architecture. The Planning agent uses a simple FMECA database and results show it is capable of choosing the correct action plans when presented with different failure cases.

1. INTRODUCTION

There is currently a considerable amount of research work focused on the development of increasingly autonomous UAV systems. The ultimate objective of this work is to provide a UAV with sufficient decision-making skills so that the operator will only need to assign a mission to it and the UAV will be able to perform it entirely, without the need of assistance from a pilot. Such capability can bring several advantages: for example, this means that a single operator can effectively control a team of UAVs being able to focus on mission management rather than on normal piloting tasks. Another example might be the case of personal UAVs for soldiers, where increased autonomy means a significant reduction in the skill level needed to pilot the UAV and a reduction in the control equipment needed.

The challenges of autonomous UAV flight become particularly difficult to tackle for civil applications. Civil missions such as global monitoring of environment and security, for example, can only be achieved if UAVs are able to fly seamlessly amongst other air traffic within national or international airspace (UAV Task Force, 2004). Furthermore, this capability has to be proven and certified, and this means that the UAV control system has to satisfy the very strict requirements typical of civil aeronautical regulations, such as the DO-178B standard (RTCA, 1992) for software development.

In order to open up opportunities for safe and routine use of UAVs in non-segregated air space by addressing specific regulatory and technological issues, ASTRAEA (Autonomous Systems Technology Related Airbone Evaluation and Assessment) was launched in 2006 as a key element of the National Aerospace Technology Strategy. ASTRAEA is a £32 million civil programme led by an industrial consortium incorporating Agent Oriented Software, BAE Systems, EADS, Flight Refuelling, QinetiQ, Rolls-Royce and Thales UK, working with leading academics and supported by investment from the DTI, Welsh Assembly Government, Scottish Enterprise and regional development agencies covering the North West, South East and South West of England.

The programme has been split into several sub-projects, each addressing the challenges of autonomy at different levels or for different sub-systems. In this light, Rolls-Royce and the Rolls-Royce University Technology Centre (UTC) in Control & Systems Engineering at the University of Sheffield are working cooperatively to develop technologies which will address the UAV autonomy issues related to propulsion and power generation systems (and particularly Gas-Turbine Engines). The objective is to improve the Diagnostic and Prognostic capabilities of the Full Authority Digital Engine Controller (FADEC), and to develop a computer system able to replicate typical pilot reactions to fault occurrence.

2. INTELLIGENT AGENT TECHNOLOGY

One of the main advances in computer science and artificial intelligence during the last two decades has been the introduction of the concept of Intelligent Agent (IA). Intelligent agents are a new paradigm in the development of software applications (Jennings et al, 1998) and are designed to address the need for flexible and autonomous computer systems.

This technology is still at quite an early stage; it has been exploited thoroughly in certain areas of application (like internet search engines), but its use in other areas of software engineering is restricted at best. In fact, even agreement on the definition of IA is not universally accepted among computer scientists. A popular definition, which we take as our own point of view, is that "an Agent is a computer system situated in some environment, and that is capable of *autonomous* action in this environment in order to meet its design objectives" (Wooldridge, 1999). Furthermore, we can say that an *Intelligent* Agent is one that is capable of *flexible* autonomous action, where flexible implies *reactivity* (ability to understand the environment and react to its changes), *pro-activeness* (goal-oriented behaviour) and *social ability* (ability to interact with other agents).

These characteristics mean that IAs are potentially a very good solution for the development of complex intelligent systems, which often must be able to replicate the reasoning process of a human being under severe real-time constraints. Computer systems are obviously able to react much faster than the human brain in simple and predictable situations, but the ability of the human brain to confront unexpected situations, change its decisions after a re-assessment, perform abstract reasoning and learn from experience remains unchallenged. IAs constitute a possible approach to narrow this gap, with the ultimate objective of building computer systems combining the fast reaction times of modern hardware with the skills typical of human reasoning.

Even though consistent advances have been made in the recent years, expertise in designing and building agent applications is still underdeveloped. There are several support tools for building IA applications but often they are over-specific and addressing only a restricted range of problems. We can identify two main trends in the development of IA tools: on one side there are "strong AI" systems, which represent the computerization of cognitive modelling theories; on the other side, certain IA tools are just an expansion of object-oriented programming languages (such as Java), introducing Agent classes so that IA concepts can be applied. Both types of approach have their advantages and disadvantages, and these must be carefully evaluated when selecting what tool should be used in a particular research project.

During this project, two IA tools were identified and assessed, one for each of the categories above mentioned; the first one is JACK™, which is an agent-oriented expansion of Java; the second one is Soar, which is instead built around a specific cognitive architecture. A final decision was made to choose Soar as our main IA development tool. This was mainly due to two reasons: firstly, because this allows us to explore the potential of cognitive modelling tools for control applications, and secondly because of issues regarding the certification possibilities of Java-based software. In fact, Java-based platforms for Safety-Critical systems such as the PERC platform from Aonix are now available, but the integration with Java-based IA tools is likely to prove difficult.

Soar provides a robust architecture for building complex human behaviour models and intelligent systems that use large amounts of knowledge (Soar Technology Inc., 2002). At a high level of abstraction, it uses a standard information processing model including a processor, memory store, and peripheral components for interaction with the outside world. At a low level of abstraction, Soar uses a Perceive-Decide-Act cycle to sample the current state of the world, make knowledge-rich decisions in the service of explicit goals, and perform goal-directed actions to change the world in intelligent ways. The distinguishing features of Soar are: parallel and associative memory, belief maintenance, preference-based deliberation, automatic sub-goaling, goal decomposition and adaptation via generalization of experience.

A Soar agent is based on its *production* rules; these represent long-term knowledge and are practically the program code for the agent. Production rules are in the form of *if-then* statements, where an action is performed only if the conditions are met. When the conditions of a production are met, the production is said to *fire*; as Soar treats all productions as being tested in parallel, several productions can fire at once, and this can happen at different levels of abstraction, giving the Soar agent natural pro-active behaviour (the agent is inherently aware whether the conditions to apply certain production rules are still valid). Short-term knowledge is instead constituted by external input, and appropriate functions must be developed to interface the Soar agent with its environment.

We will now look at how the Soar tool can be used in the field of Gas-Turbine Engine (GTE) Health Management. In this case, we aim to develop a system able to correctly manage the running operation of a GTE, both in a fault-less and in a faulty condition. This system must optimize the operation when no fault is detected, and apply appropriate reversionary and investigative action plans when a fault has occurred.

3. ENGINE HEALTH MANAGEMENT SYSTEM

In the last two decades, avionic systems have become more and more important in an aircraft. The extraordinary advances in electronics allowed the introduction of complex control hardware for all the on-board systems. GTEs are not an exception and the introduction of Full Authority Digital Engine Controllers (FADEC) brings several benefits to their operation (Harris et al, 2000):

- longer life guarantees;
- improved operability including fast handling capability and minimum pilot intervention even after incidents;
- ease of maintenance due to the availability of performance data and fault diagnostics
- increased integration with airframe systems;
- lower life cycle costs

Modern FADECs usually include Engine Health Monitoring (EHM) capabilities; this means that the FADEC is connected to a number of sensors and logs all data coming from these. The FADEC can respond quickly to critical conditions such as flameouts or stalls, but most of the EHM data is used only for ground maintenance.

With this work, we aim to use EHM data to adjust engine behaviour during flight, so that the effects of a fault can be mitigated. This is a complex task, requiring:

- correctly identifying a fault
- understanding the effects both at engine level and at platform level
- devising appropriate reversionary action plans

Also, we aim to optimize engine operation in order to achieve lower-priority objectives such as maximising engine life, minimising fuel consumption and maximising engine response. All of this has to be accomplished under severe real-time constraints, as on-board hardware is usually resource-constrained and the system has to continuously monitor data from all sensors. Also, if the system is to be given real autonomy, it becomes safety-critical, adding severe requirements to ensure operability.

The approach we used is based on a distributed architecture. The system consists of a general Intelligent Agent architecture, which is needed to interface the different types of technology used to provide the functionality. The use of the IA paradigm allowed encapsulation of different functions such as Fault Detection, Fault Isolation and Planning in different agents. The agents can then interact through the architecture, with the general behaviour of the system emerging from their interoperation.

A key objective in assembling the agent architecture has been the possibility to integrate different types of Intelligent Systems technology. In fact, the various tasks that the system has to perform are best obtained from a coherent mix of diverse technologies. In particular, at present the following technologies have been integrated in the architecture:

- Digital Signal Processing is used for Fault Detection
- Case-Based Reasoning (CBR) is used for Fault Isolation
- Fuzzy Logic is used to assess the effects of a fault at platform level (where by platform we intend the platform which is hosting the engine, e.g. the UAV)
- Intelligent Agent tools are used for reversionary action Planning

In terms of data flow, the obvious start is the FADEC, which collects engine sensor data; additional data is coming from the platform (environmental and mission data). Raw sensor data is digitally processed, partly by the FADEC and partly by our system, in order to achieve Fault Detection.

Fault Isolation is accomplished using the Intelligent Fault Isolation System (IFIS), which is a tool also developed at Sheffield University (Mills et al, 2006). This system is based on CBR technology and is designed to fuse data from different sources, extracting “hidden information” already present in this data but difficult to interpret without contextualising the symptom using other sets of data not directly related to it (i.e., data from an additional sensor might confirm or discard the fault detected by a main sensor).

Output from IFIS is fed into Fuzzy-Logic-based algorithms that evaluate the effects of a fault. Engine performance is evaluated using a series of engine performance parameters. A Fuzzy Inference System relates faults to a numerical estimation of their effect, calculated in terms of a reduction to these parameters. The criticality of a fault is also computed, by assigning a criticality degree to each of the possible effects to engine performance.

Finally, all of this data is used by the Planning agent to select the correct reversionary action plans. The agent considers both EHM data and situational awareness data, in order to make contextual decisions: we do not want the system to reduce thrust during take-off because it detected a minor fault! In fact, the system is built to always operate under the authorization limits provided by the platform, so this would not be possible unless it is requested by the platform itself, but it is also important to avoid requesting to the platform a reduction in thrust when this is clearly not reasonable. The Planning agent also serves as an intelligent thrust demand optimization tool, which is useful during normal engine running condition (no fault present). In this case, the system can assume several different behaviours, such as maximising engine life, minimising fuel consumption or maximising engine response. The platform can impose a preferred behaviour, but the decision on which is the best behaviour is also influenced by EHM data.

We will now concentrate on describing the structure of the Planning agent in Section 4, while Section 5 will present the tests performed on the agent. The other aspects of the architecture are described in more detail in a separate publication (Gunetti et al, 2007).

4. THE PLANNING AGENT

Incorporating Soar technology with the Intelligent Agent architecture involved realizing an appropriate interface. As the architecture is built using Matlab/Simulink software, this task meant writing an S-Function encapsulating the Soar agents.

The S-Function is a custom-built Simulink block, which incorporates the code for the Soar kernel to effectively run an instance of Soar inside a Simulink model. The interface is designed to be general, so that it can be re-used with minimum changes. In fact, the user interface allows to freely choose what production rules the agent should load, the agent name and the port number (used, for example, to connect the Soar Debugger tool). The Input/Output structures are instead hard-coded, meaning that to change these it is necessary to change the S-Function code and recompile it. The Soar/Simulink is now being optimized and a basic version should be included in the next main Soar release.

The S-Function encapsulates a single Soar agent with the rules specified in a file. In order to build a Soar agent, one must write these production rules in a text file (normally using the VisualSoar IDE). To correctly write the rules, it is imperative to carefully think about how to organize the agent’s execution cycles. The Soar programmer must understand how to divide the problems the agent must face into different categories, and plan *a priori* the flow of the decision process.

The Planning agent is a decision-maker, taking Fault Diagnosis and situational awareness data as inputs and outputting an optimized thrust demand together with reversionary and investigative actions. The symptoms, faults and reversionary actions which have been embedded in the system and in the Planning agent are derived from a sample Failure Mode, Effects and Criticality Analysis (FMECA) database provided by Rolls-Royce, who are an industrial partner in the project. It is important to stress that this work is intended to demonstrate the potential for such technology; therefore, the FMECA database is simplified but representative of the problem domain. This allows focus to be placed on working directly on the technology rather than on the implementation of a detailed database. It is expected that following the demonstration of the suitability of this technology in the field of GTE Health Management, a larger and more realistic database will be implemented. The Agent architecture and Planning agent are built with this in mind, so that implementation of a new database will be as seamless as possible.

Technically speaking, the Soar Planning agent is built using the abstraction of several levels of decision-making. Figure 1 shows the operator hierarchy adopted in the Planning agent. The main organizational division is between the two cases of:

- No fault detected (running-normal)
- Fault-detected

Soar continuously checks input in order to find out whether a fault is present. If no fault is detected, it enters the "Running-Normal" state, whereas when a fault is present it switches to the state of "Fault-detected".

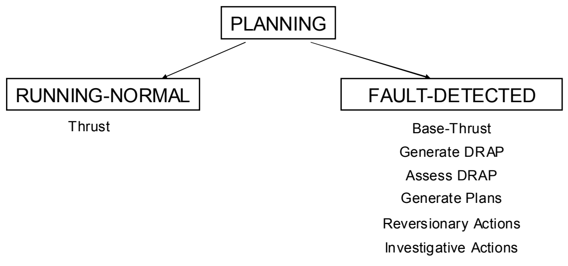

Figure 1 – Planning Agent organizational diagram

In the first case, the agent must only determine the thrust level to be applied, within the limits provided by the platform. The decision on thrust level is based on input coming from the platform (mission phase advisory and environmental condition) and from its own long-term knowledge (for example, the agent can have long-term knowledge of optimal thrust ranges). An interesting possibility is using the Soar learning system (called Chunking) to improve this aspect. The controller might analyze sensor data and calculate what the optimal thrust ranges are, then this could be "learnt" using the chunking technique (which basically involves adding a new production rule to the predetermined set). There is a drawback though; in fact such a characteristic would drastically decrease the predictability of the system, especially if new rules are to be added automatically. This would certainly lead to certification issues. For this reason, the learning capabilities of Soar have not been used at this stage of the project.

As soon as a fault is detected, the agent enters the other main state, which is the "Fault-Detected" state; in this situation the agent goes through a series of tests in order to determine appropriate action. At first, it calculates a "Base-thrust", which is basically repeating the same process of "Running-Normal" but with (possibly) different parameters. This is needed to have a starting point for the next operators. The next decision phases are the ones during which a Draft Reversionary Action Plan (DRAP) is generated and then assessed against the action scope provided by the platform. If the DRAP is found to be falling out-of-scope, additional plans are generated, up to a number which is dependant on fault-criticality. Finally, additional reversionary and investigative action (such as requesting return to base or running additional tests) are assessed and the agent evaluates whether these are appropriate or not. When a complete plan is ready, filled with all details about programmed reversionary and investigative action, output is sent back to the agent architecture, which verifies that the command is consistent and then forwards it to the FADEC. In order to make it more predictable, plan generation has been designed as a scheduled activity; the generation of a plan will always go through the same series of decisions, in the same order.

This main set of rules is complemented by additional productions such as the state elaboration rules, which create useful abstractions of input data, and the I/O management rules, which organize I/O data and interface with the I/O functions.

Having described the structure and organization of the Planning agent, we now proceed to test it in a simulated environment, based around the IA architecture and complemented with a GTE model.

5. TESTS AND RESULTS

The Planning agent was tested in a series of simulations where its behaviour under pre-determined conditions was recorded. The tests saw the use of the entire agent architecture with all the tools currently implemented; these had to be complemented with a Fault Injection tool, visualization tools and a GTE model which is used to verify how the entire system behaves when controlled by the Planning agent.

The sample FMECA database in this test is constituted by a set of fourteen symptoms, which are mapped to six different faults. Reversionary and Investigative actions are related directly to the faults, but are also dependant on the current situation (i.e. actions are contextualized, so, for example, the system does not try to reduce thrust during take-off, even if the occurrence of a fault would suggest it from the engine point of view). The plans are devised in real-time, by progressively assessing the need for actions. This is a major difference compared to more conventional technologies, employing traditional techniques such as look-up tables. With these, the system always reacts in a pre-specified manner, and each possible input configuration is directly mapped to an appropriate reaction. With IAs, we instead generate a plan by sequentially assessing the need for each action.

The testing methodology involved running simulations in a Simulink environment, during which the entire architecture was in execution with simulated input. The inputs were mainly from the Fault Injection tool, but also from dedicated Simulink blocks designed to simulate the presence of the platform with its thrust requirements. A number of scenarios were identified, consisting of different environmental conditions and fault cases. Output from the Planning agent was then compared with output from a more conventional algorithm performing the same tasks and finally fed into the engine model to determine the realistic effects of the agent's decisions. The conventional algorithm is based on look-up tables derived from empirical data and the results of the comparison are omitted due to space reasons.

We will now present part of the results of the tests; these are obtained from a scenario where a standard flight condition is seeded with different faults. The simulated platform is requiring a thrust level within 40 to 60 percent of maximum thrust, and these limits cannot be exceeded by the system without authorization from the platform. Results show that the system correctly identifies the faults and takes appropriate reversionary and investigative actions. Results are shown in three graphs, depicting several aspects of the agent's decisions.

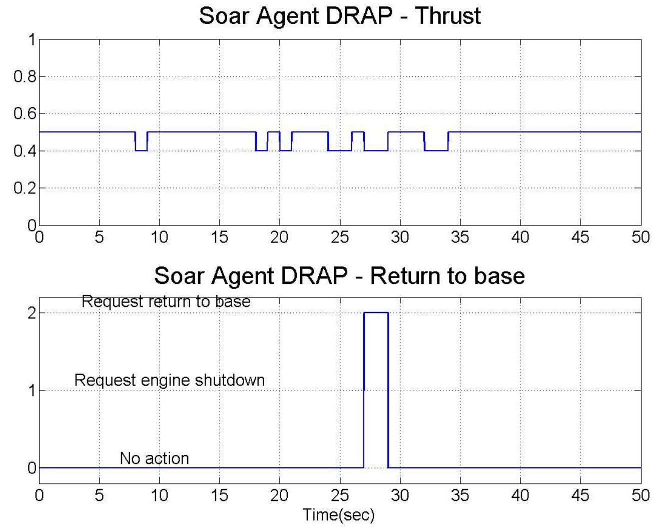

Figure 2 – Reversionary action plans

Figure 2 shows what the reversionary actions are; these practically consist of a reduction in thrust level, which is always within the action scope provided by the platform. In case the agent decides for a thrust level which is out of the action scope, the selected level will be the closest one falling within the scope. The platform is informed that the optimal plan from the engine point of view is out-of-scope but advisable in terms of engine performance and health. It is important to understand here that our system plans thrust levels and all actions in order to pursue the "best interest" for the GTE; this may significantly differ from the requirements of the platform, which obviously have a higher priority.

In the first graph we can notice the time plot for thrust demand, while the second graph depicts when extreme reversionary action is taken; in this case, for a very critical fault the system advises the platform to return to the base as soon as possible; this is the case when there can be no guarantee that the engine can complete the mission.

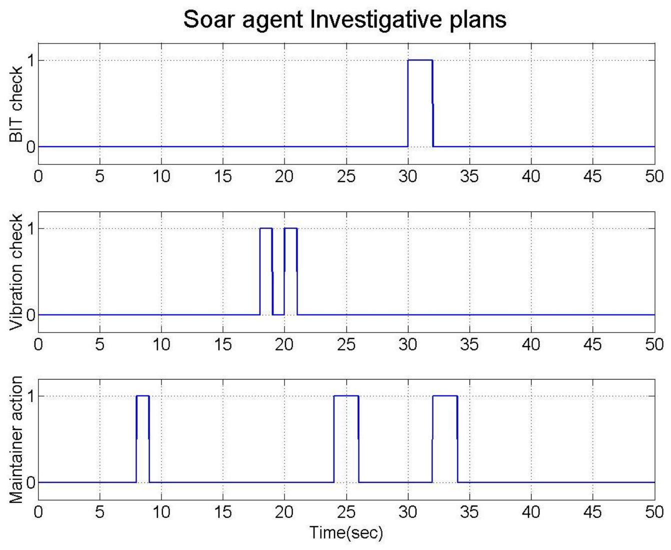

Figure 3 – Investigative action plans

Figure 3 shows the possible Investigative actions and their plots. The agent performs appropriate investigative action only if this is not interfering with reversionary action. Investigative actions include performing additional tests (BIT check), performing a detailed vibration analysis and signalling the necessity of maintenance when on-ground.

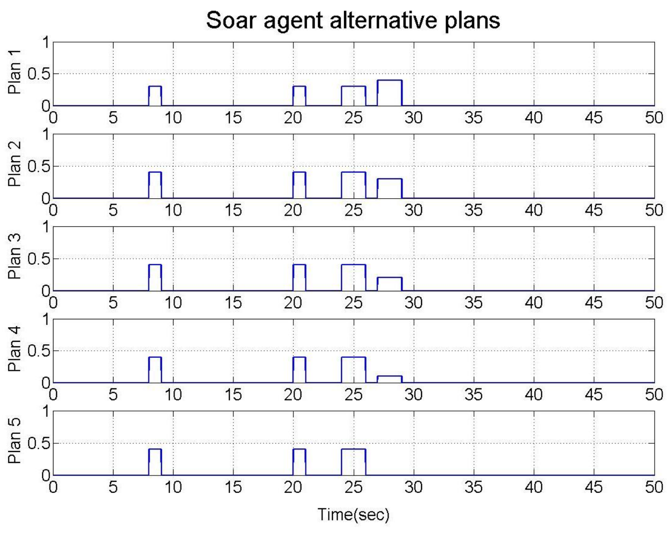

Figure 4 – Alternate out-of-scope plans

Finally, Figure 4 shows the alternate plans which are generated by the agent. This happens only when the optimal plan is not falling within the action scope. In this case, the agent applies an immediate change of thrust falling within the action scope, but also sends an advice message to the platform to inform it that more drastic action is needed. Depending on the criticality of the fault (which is numerically estimated by previous algorithms), a different number of plans is presented to the platform, with more critical faults allowing a larger number of plans, up to a maximum of five plans.

These results show that the Planning agent is capable of applying appropriate Reversionary and Investigative action plans when presented with different fault situations. Comparison with results from conventional technologies demonstrates that the Soar agent is able to give a similar level of performance, if not better. We must consider here that the implemented database is simpler than a realistic one. Performance may vary consistently with a more realistic database, but it is expected that it will still benefit from the use of IAs.

The Planning agent also implements thrust optimization capabilities which are very complex to recreate using traditional techniques. These were very simple to model using the Soar language, as Soar is very well suited to performing symbolic reasoning tasks. In fact, as stated before, the Soar learning capabilities make the development of such an adaptive system much simpler when compared to other technologies.

6. CONCLUSIONS

In this paper we have presented a novel approach to Gas-Turbine Engine Control and Health Management. The approach focuses on the use of the Intelligent Agent paradigm to model the various software components of the system. Several types of technology were integrated by encapsulating them in Intelligent Agents, which formed a distributed IA architecture.

The main function of the system was choosing appropriate reversionary and investigative action plans. This was performed by an agent built using the Soar IA tool. The paper explains how the Soar agent is integrated with the rest of the system, details the modelling logic behind the agent and presents simulation tests.

Tests demonstrated that this approach is practicable; in fact, the use of IA tools allowed not only the desired behaviour from the system to be obtained, but also implementation of more advanced features such as thrust optimization. From this, we estimate that the characteristics of Soar make it suitable to implement Adaptive Scheduling in the system. This is a very complex feature which is part of the requirements for the ASTRAEA project, and Soar technology should allow this to be put it into practice with relative ease compared to more conventional technologies.

Four prospective areas of research have been identified:

- Implementation of multiple agents;
- Implementation of a larger and more detailed behaviour model;
- Experiments on real-time implementations of Soar
- Experiments on the use of Soar learning capabilities

Future work will address these issues, but will also be dedicated to exploring the possibilities of using Soar agents for general UAV control. In fact, a lot of the research work done is meant to be easily portable to an autonomous UAV control unit. It is felt that the potential of Soar agents for high-level mission management is great, and our efforts will look at a practical demonstration of this capability.

7. ACKNOWLEDGEMENT

The authors acknowledge the support of the DTI, the ASTRAEA consortium and Rolls-Royce Plc in this work.

REFERENCES

Gunetti P., Mills A. and Thompson H., “A distributed Intelligent Agent architecture for Gas-Turbine Engine Health Management”, *46th AIAA Aerospace Sciences Meeting and Exhibit*, 7 – 10 January 2008, Reno, NV (to be published)

Harris P., Swain B., Webb K., “The Control and Monitoring System for the Adour 900”, *Aircraft Engineering and Aerospace Technology, Volume 72, Number 6*, 2000

Jennings N., Wooldridge M., “Applications of Intelligent Agents”, in *“Agent Technology: Foundation, Applications and Markets”*, Springer, 1998

Mills A., Tanner G., Thompson H. and Fleming P., “On-Wing Decision Support for Aero-Engine Line Replaceable Unit Fault Isolation”, *International Symposium on Air Breathing Engines*, ISABE-2007-1290

Radio Technical Commission for Aeronautics (RTCA), “Software Considerations in Airborne Systems and Equipment Certification”, published by RTCA, Inc. 1992

Soar Technology Inc, “Soar – An overview”, © 2002

UAV Task Force, “The Joint JAA/EUROCONTROL Initiative on UAVs”, UAV Task Force Final Report, 2004.

Wooldridge W., “Intelligent Agents”, in *“Multi-Agent Systems: a modern approach to distributed artificial intelligence”,* the MIT Press, 1999