A distributed Intelligent Agent Architecture for Gas-Turbine Engine Health Management

Paolo Gunetti[1], Andrew Mills[2] and Haydn Thompson[3]
University of Sheffield, Sheffield, South Yorkshire, S1 3JD, United Kingdom

Control and Health Monitoring of complex systems such as Gas-Turbine Engines can potentially receive great benefits from the use of advanced software technologies. However, techniques such as Intelligent Agents, Neural Networks and Genetic Algorithms are predominately designed to optimally perform specific functions, while the rest of the functionality is better achieved using conventional techniques. In this paper, we describe the development of a simulation architecture for Gas-Turbine Engine Control and Health Monitoring. This architecture allows integration of advanced software technologies with conventional modelling techniques. The architecture is then used in the implementation of a complete Health Monitoring system that utilises Case-Based Reasoning to achieve Fault Isolation and Intelligent Agents to achieve Fault Mitigation.

I. Introduction

THE last two decades have seen a dramatic increase in the applications of Artificial Intelligence (AI) techniques. In fact, after the great theoretical achievements of the 1960s, AI research suffered serious cutbacks both in terms of interest and funding. This was partly due to the inadequacy of available hardware. With the extraordinary evolution of hardware and computing power that began in the 1980s and the diffusion of such hardware, research in AI finally became mainstream again, and the ideas originating from the 1960s found many applications which were not possible at the time [1].

We can identify four main types of techniques that have been developed:

1) Formal AI methodologies, which include the relatively new field of Intelligent Agents (IAs)
2) Fuzzy Logic
3) Artificial Neural Networks (ANNs)
4) Genetic Algorithms (GAs)

These techniques can be grouped together into the single definition of "soft computing". In the last two decades, they have been used in a very broad range of applications; for example Intelligent Agents are widely used in the World Wide Web, while Fuzzy Logic systems found application in several consumer electronic products. Altogether, soft computing techniques have become increasingly popular for Control applications, where their capabilities can bring great advantages in terms of optimization of functionality.

Due to its conservative nature, the aerospace industry has been quite resilient to the introduction of soft computing. While a large amount of research has been undertaken and specific applications developed, especially in the military field, wide-scale application in the civilian field is lagging behind.

An area where soft computing could potentially be exploited with good results is aeronautic propulsion. Gas-Turbine Engines (GTEs) are driving a significant amount of research focused on increasing reliability and improving functionality. The now wide-spread adoption of Full Authority Digital Engine Controllers (FADECs) has brought several advantages to GTE operation and opened the way for the use of soft computing techniques in this field. With respect to the two main focuses, soft computing could be useful in:

- Increasing reliability by improvement of current Fault Detection and Isolation (FDI) methodologies, optimization of Maintenance processes and definition of new methods to analyse GTE usage data
- Improving functionality through fuel efficiency maximisation and optimization of control algorithms

[1] Research Assistant, ACSE Department, p.gunetti@sheffield.ac.uk, AIAA Member.
[2] Research Associate, ACSE Department, a.r.mills@sheffield.ac.uk.
[3] Professor, ACSE Department, h.thompson@sheffield.ac.uk, AIAA Member.

A key point in using soft computing techniques is that they are best suited to very specific tasks but it may be desirable to compliment these techniques with "conventional" modelling and data management techniques such as look-up tables and matrix logic in order to be used in a real system.

In this paper, we present a software architecture that has been developed to implement and test various soft computing solutions that can be applied to GTE Control and Health Management. The architecture is based on a design that allows different types of algorithms to be used for different stages of the Health Management process, such as Fault Detection, Fault Isolation, Fault Diagnosis, System-Level Prognosis and Fault Mitigation. The architecture also provides a simulation environment so that the implemented algorithms and their interaction can be evaluated.

The paper is structured into four main sections. Section II gives an overview of the techniques that we would like to use and the advantages they bring. Section III describes the architecture in detail. Sections IV and V present examples of integration of soft computing techniques. In section IV, Case-Based Reasoning is used for Fault Isolation, while in section V Intelligent Agents are used for Fault Mitigation.

II. Soft computing techniques

There are many examples regarding the use of soft computing techniques in the Gas-Turbine Engine Health Management field. In this section, a brief overview of these applications will be presented. First, Artificial Intelligence approaches will be presented, with an emphasis on rule-based and case-based systems. Next, Fuzzy Logic and Neural Networks will be discussed, and finally Genetic Algorithms.

A. Formal AI techniques

Formal AI technology has developed in many directions, but the most common application is represented by *Knowledge-based* or *Expert* Systems. These systems provide a way to represent and use generic knowledge of some type within a computer system.

The foundation of an expert system is a knowledge base, which is built to be used by an inference engine to retrieve information regarding a specific subject [2]. Several types of expert systems have been developed, such as rule-based, model-based and case-based. These have found several applications to GTE Health Management, mostly in the Fault Detection and Fault Isolation processes. Also, expert systems have been combined with other techniques such as fuzzy logic, probability theory and belief functions in order to deal with problems with uncertainty. Some of the most recent applications combine knowledge-based system with Bayesian belief networks [3].

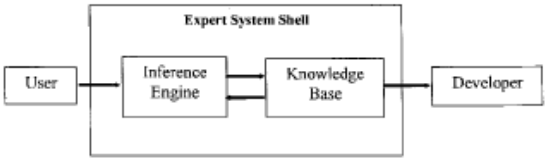

Figure 1. Configuration of an Expert System

In section IV of this paper, a software system using Case-Based Reasoning in order to achieve Fault Isolation will be introduced and integrated within the GTE Health Monitoring architecture. In section V, the integration of an Intelligent Agent will be described. Intelligent Agents (IAs) are a relatively new branch of AI technologies that can be considered to be one of the latest evolutions of formal AI techniques.

B. Fuzzy Logic

Fuzzy logic could be described as a method to formalize the human capability of imprecise reasoning. Such reasoning represents the human ability to reason approximately and judge under uncertainty. It provides a system of non-linear mapping from an input vector to a scalar output. A typical fuzzy logic system involves fuzzification, fuzzy inference (based on a set of rules) and defuzzification [2].

Fuzzy logic can be used for gas turbine diagnostics, usually by combining it with other technologies. Fuzzy Logic and Fuzzy Inference Systems are commonly used in Control applications in order to build robust systems that can deal with uncertainty situations and imprecise data. This is also true in the field of GTE Health Management: while fuzzy logic itself is usually not sufficient to model the entire system behaviour, it is often a necessary complement for other techniques.

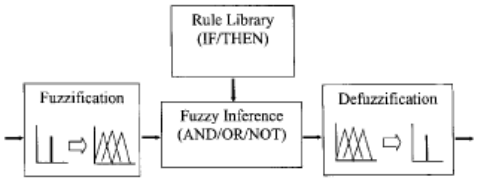

Figure 2. Configuration of a Fuzzy Inference System

C. Artificial Neural Networks

Neural networks are massively distributed processors designed to mimic the human brain. These systems show the particular ability of being able to store experimental information and making it available for use.

ANNs were first studied in the 1960s, but at the time computer technologies did not allow real world application. Following the advancement of computing technology, interest in ANNs was renewed. ANNs are now widely used in Control applications.

An ANN needs to be trained using experimental data and some type of training algorithm. After training has been completed, it can be used to model the relationship between the experimental results and the relative inputs. Since the learnt information is actually stored within the changing values of the weights that describe the relationship between neurons (basic components of a neural network), it is actually difficult to understand the meaning and justification for the stored information.

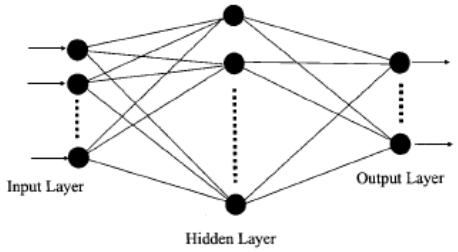

Figure 3. Configuration of a Feed-Forward Back-Propagation Neural Network

Research on the application of ANN in the GTE Health Management has lead to several interesting developments [6], but real-world application of these is not wide-spread, often due to certification difficulties.

D. Genetic Algorithms

Genetic Algorithms are a searching and optimization technique. They involve the definition of an imprecise solution to a problem, from which alternative solutions can be generated by introducing random variations that can be evaluated through some evaluation parameter. The process is iterated until an optimal solution is reached.

Although GAs are mainly an optimization tool, they have found some applications in GTE Health Management too, notably in the optimization of fault diagnosis in the presence of measurement noise and biases [4]

III. The architecture

The studies listed in section II detailed the use of various types of techniques to achieve specific functions typical of Health Management. Instead, in this study we want to build a complete Health Management system that performs all of the functions that can be expected by such a system, including Fault Diagnosis, Fault Isolation, Prognosis of Fault Outcomes, and Fault Mitigation. This involves finding a way to integrate diverse functionality and technology. In this section, we will describe the architecture that has been developed in order to address these issues.

Logically, the first step in building the architecture was defining a set of requirements. The architecture must be designed to:

- Perform a complete Health Management task
- Ease the integration of technologies which are very different in nature
- Include a GTE model
- Give the possibility to evaluate software components based on soft-computing techniques

The architecture is developed using the Matlab/Simulink software package. This choice was natural for two reasons; firstly, Simulink fulfils the requirements and secondly, a consistent amount of research on soft computing actually uses Matlab/Simulink [5].

The approach we used involved the development of sample functionality covering the whole Health Management system using conventional modelling techniques (in particular, look-up tables and matrix logic). These sample algorithms provide later the possibility to compare results with the more advanced algorithms based on soft-computing techniques. To achieve this, the interfaces between the different stages of the Health Management process must be clearly defined.

It is important to define what we mean by a complete Health Management task. With the help of Rolls-Royce, the industrial partner in this project, a set of requirements for the Health Management task was defined. These requirements state that a complete Health Management system must be able to:

- Correctly detect and identify faults
- Assess the effect which a fault has both at system-level (the engine) and platform-level (the aircraft)
- Assess the need for further investigation in the case of uncertain faults

- Choose appropriate reversionary action in order to mitigate the effects of faults
- Optimize engine operation so that different "behaviours" can be applied (i.e. maximise range, maximise performance, maximise engine life)

The requirements were then translated into a Simulink model (Figure 4). The model identifies three main sub-tasks within the Health Management task; these are Symptom Prognosis and Fault Diagnosis (SPFD, yellow block), System Health Assessment (SHA, orange block) and Planning and Authorization (PA, light blue block).

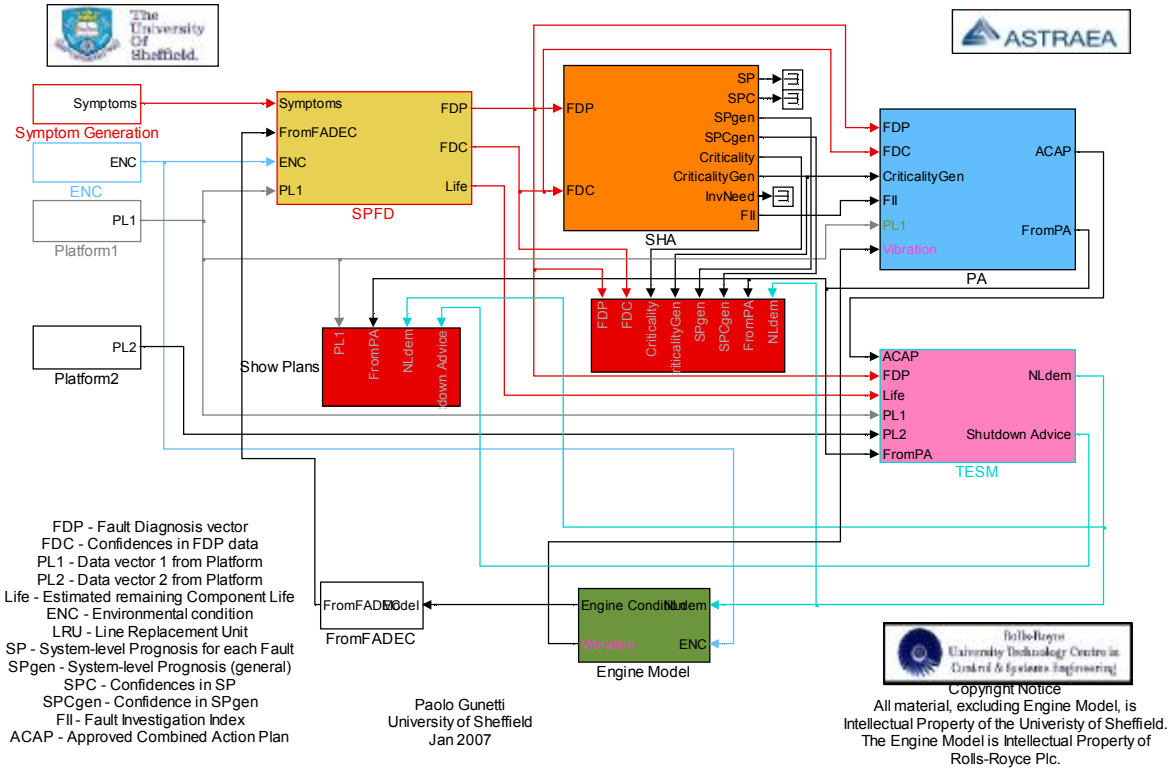

Figure 4. Main level of Simulink model of the Health Management architecture

As a result of the well-defined interfaces, each sub-task can be implemented with a variety of soft computing approaches (see section II) or by simple look-up tables. The different approaches for each sub-task can be compared using the simulation tools included in the architecture. We will now describe each of the sub-tasks in detail, explaining the functions performed, the interfaces needed and the preliminary algorithms used.

A. SPFD area (figure 5)

This sub-task performs the Fault Detection and Fault Isolation functions. Fault Detection consists of the identification of anomalies within data coming from sensors and from the FADEC, such as the identification of specific vibration signatures or

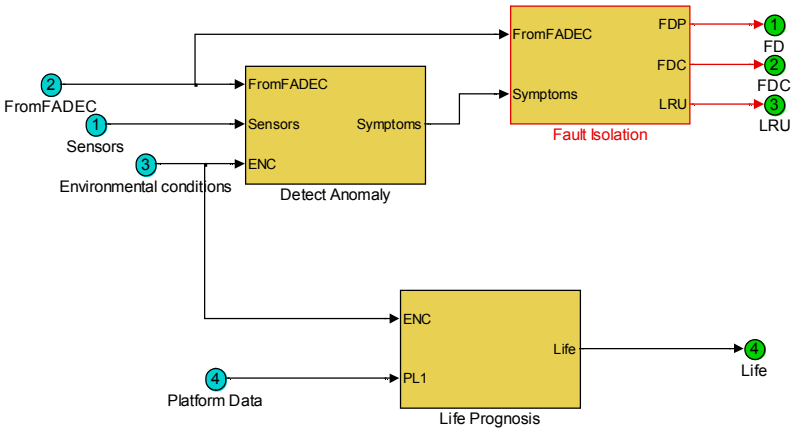

Figure 5. Functions within SPFD area

degraded engine performance. The "Detect Anomaly" block converts raw sensor data into a set of binary indicators that indicate the presence of specific Symptoms. Every different degraded condition is related to a different Symptom indicator. Symptoms are then fed into a Fault Isolation System, which combines them using data from a Failure Modes, Effects and Criticality Analysis (FMECA) database in order to isolate the fault. The output from this consists in two vectors indicating the diagnosis of specific fault conditions and the confidence in this diagnosis. The Fault Isolation process also provides an indication of the Line Replacement Units (LRUs) that the detected faults affect.

Within SPFD is also a system which is used to determine a Prognosis for the remaining Component Life. This function complements the Fault Detection and Fault Isolation tasks and provides additional data which can be used in the subsequent tasks.

These three blocks are initially implemented in a discrete rule system. In section IV, the Fault Isolation block will be replaced by a Case Base Reasoning isolation tool.

B. SHA area (figure 6)

This sub-task is dedicated to the generation of additional diagnosis and prognosis data, which is considered to be useful in the generation of Fault Mitigation plans. Three types of data are generated: a prognosis for the effects of faults at system-level and at platform-level, an assessment of the criticality level for faults and an indication for the need to investigate faults which have an uncertainty level.

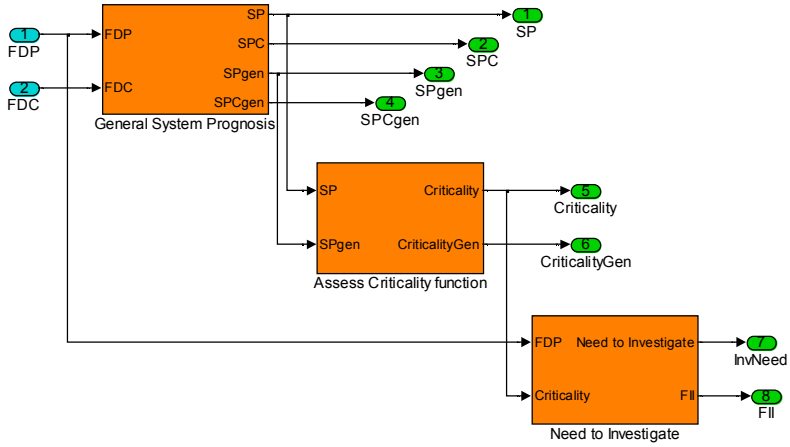

Figure 6. Functions within SHA area

The System-Level Prognosis block takes input from the SPFD area regarding what faults are detected and the confidence in the detection and outputs the effects of faults in terms of platform-level performance. Platform-level performance is evaluated by indicators that describe effects such as efficiency loss, available thrust loss or mechanical break-ups in progress. Confidences in these prognoses are also calculated.

Criticality is defined as a numerical indicator of the severity of a fault. Five levels of criticality are defined, ranging from level 5 (No effect) to level 1 (Catastrophic failure). The Assess Criticality block performs this estimation and this data is then combined with Fault Diagnosis data in the "Need to Investigate" block, where uncertainty in fault detection will lead to the indication that Investigative action is needed.

Again, all of these functions are implemented using matrix-based modelling and look-up tables. As part of the testing for the architecture, the Assess Criticality block was substituted with a block based on a Fuzzy Inference System (FIS). While integration was seamless, the use of this technology did not bring any advantages to this function, due to its simplicity. It is however necessary to point out that this test was motivated by the need to verify that integration of Fuzzy Logic techniques within the architecture could be easily accomplished. This is achieved using the Fuzzy Logic Toolbox, which is an optional component of the Matlab/Simulink package.

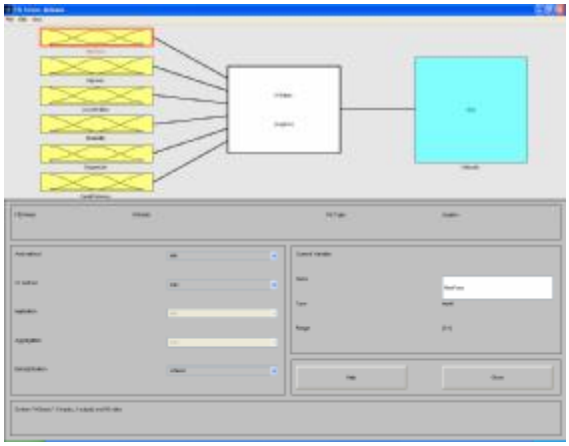

Figure 7. Fuzzy Inference System for Assess Criticality function

The Toolbox allows seamless integration of Mamdani-type and Sugeno-type FIS within Simulink models.

C. PA area (figure 8)

This sub-task is dedicated to performing Fault Mitigation and Investigation. Fault Mitigation involves the generation of Reversionary Action plans, which basically consist in an indication to the platform (the pilot or the supervisory authority in case of an autonomous UAV) regarding limits to be placed on the thrust level required from the engine, complemented by a list of advisories regarding aircraft behaviour (such as "minimise changes in thrust" or "avoid negative G-loads"). Fault Investigation is instead focused on plans that have a possibility of improving the confidence level in an uncertain case of Fault Diagnosis. Such plans can be of two types: commands to change the thrust level to specific settings or execution of auxiliary test routines. Both Reversionary and Investigative Action plans are then assessed to determine how they would affect platform operations and then combined in order to obtain a list of generic plans. These plans are also assessed regarding their effect on platform operations and finally sent to the platform which is the authority responsible for deciding whether a plan should be applied or not.

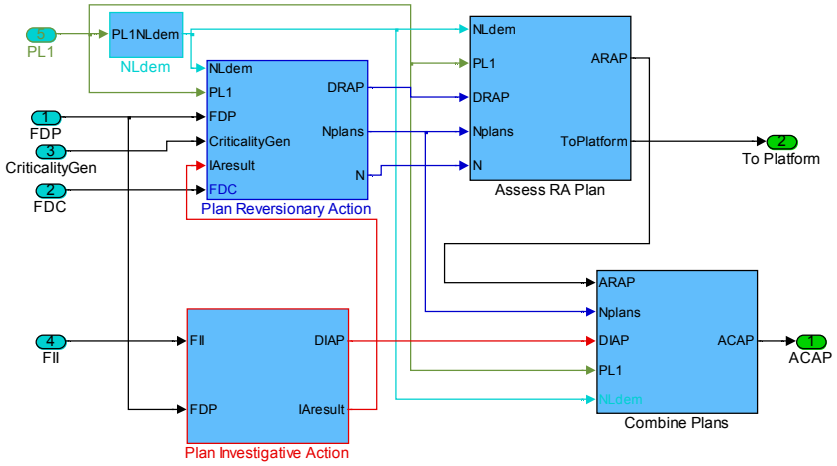

Figure 8. Functions within PA area

The basic algorithms used in the architecture use matrix-based modelling and look-up tables, as in the previous sub-tasks. In section V, the entire functionality will be recreated using Intelligent Agents.

D. Additional components

The three main areas of the architecture are complemented by additional components needed to complete the system. Looking at Figure 4, it is possible to notice:

- Platform data simulation blocks (white, with light blue or grey border); these blocks simulate input that is sent by the platform (the aircraft), including commands to the engine and environmental conditions sensor data.
- Fault injection block (white with red border); this block allows the scripted or manual injection of faults during a simulation. Faults can be injected at two different levels: as variations on sensor data (prior to "Detect Anomaly" function) or as detected Symptoms (prior to Fault Isolation function).
- Thrust, Energy and Schedule Manager block (magenta); this block is a template for the insertion of optimization functionality, but is currently unused
- Engine model (dark green); this block is a simplified GTE model that is used to evaluate the effect of actions made by the Health Management System.
- Visualization blocks (red); these blocks use the Dials & Gauges Simulink Toolbox to visualize in real-time the simulation results.

Having described the Health Management architecture, we will now proceed by portraying two examples of integration of soft computing technology. These examples will be the subject of the next two sections.

IV. IFIS integration

In this section, an 'off the shelf' software component, in this case an application using the Case Base Reasoning AI approach, is integrated into the Health Management architecture to perform one of the functions described in section III. This tool is called Intelligent Fault Isolation System (IFIS) and is thoroughly described in another paper [7]. The IFIS tool was originally designed as a Civil GTE On-Wing system capable of reliably identifying the location of an engine fault to a specific Line Replaceable Unit (LRU).

Civil GTEs have a multitude of sensors measuring temperatures, pressures, speeds, vibration levels and oil particulates. Recent engine health monitoring advances have seen the development of functional sub-systems, which perform on-engine processing of some of these measured signals with algorithmic techniques. These algorithms extract information from these measurements to produce so called features representative of some abstracted engine characteristic, and supplement more traditional self-test processes such as BITE. Individual features, the raw measurements and system self-test functions give some indication of the engine state, but need to interpreted and combined to best determine the health of the engine and isolate any failure.

In this application the IFIS system was chosen to compare against the originally devised discrete rule implementation. Whereas the discrete system will only give a result if an exact match is made, the IFIS system will produce a 'best guess' failure scenario for situations where no rule is exactly the same as the fault symptom pattern. Both methodologies use the same rule base, which contains expert or case history knowledge of how symptoms are related to faults.

The patterns of symptoms representing known failure cases are stored as algebraic rules. The main rule operators are a set of logic operations (AND, OR, NOT, etc) and a set of threshold operators (<,>, etc). The operands can be real numbers, feature values or signal measurements. The use of the threshold operators and real numbers allows a signal measurement to be converted to a Boolean true or false.

The discrete system returns a true or false based on whether the pattern stored matches the pattern received. The IFIS system outputs more information by determining by how much the symptom pattern failed.

A. IFIS Operation

IFIS constantly monitors the system input for a feature to be detected, and once this occurs a window of variable size is opened to monitor all the features that are set high during the time period and temporally log the raw sensor measurements. This set of features represents the state of the engine.

The Inference logic of IFIS calculates a closeness of fit metric against the feature inputs collected during the time window, for each failure case pattern stored in the Rule Base. The closeness of fit metric is based upon a weighted 'nearest neighbour' calculation and allows engine states not matching a specified failure case to be handled and thus allows robustness to knowledge uncertainty. The closeness of fit metric is a variant of the Total Distance Metric [8], and uses a ratio of matches to number of features. The closeness of fit scoring is performed by counting the logical result of each feature examined by the rule. These scores are parsed to select the highest scoring cases. We now have a failure case or set of failure cases which best represent the health state of the engine.

Case Description	26-32003 (L)	26-32056 (L)	Noise	Broad-band	P30
FOD ingestion	N	N	Y	M	>500
Sensor Fault	Y	Y	M	N	>300
Hamess Fault	Y	Y	N	N	<50
Combustor fault	N	N	Y	Y	>100
Gearbox	N	N	Y	Y	M

Table 1. Example failure case rule set

Typically rules will be created such as those illustrated in Table 1. The table represents the algebraic definition of the fail cases, for example the rule Sensor Fault has been created to identify a pressure sensor fault: if P30>300 AND 26-32003(L) AND 26-32056(L) AND NOT Broadband THEN TRUE. In the table, terms are defined as: 26-32003(L) & 26-32056(L) are Boolean BITE messages (EHM system 1) indicating pressure sensor faults, P30 is a sensor reading (EHM system 2), and Broadband is Boolean feature from a vibration signal processing system (EHM system 3). Thus there are 3 different fault detection subsystems contributing to the rule. The M represents a Maybe, where there is some uncertainty in the manifestation of the feature for the particular failure case.

Input	Captured value Engine
26-32003(L)	1
26-32056(L)	0
P30	360
Noise	1
Broadband	0

Table 2. Example Symptom Set for a proposed single fail case

During operation an event occurs and features are captured from the EHM subsystems during the allotted time window. The hypothesised collected values are illustrated in Table 2.

Application of these calculations, both IFIS and Discrete, to the table Fault Cases has been performed. The results are shown in Table 3 indicating the Discrete System Decision compared to that of IFIS. It is shown that IFIS can produce estimation of similarity for each of the Fail Cases. Despite no exact matches the IFIS system will return a result, unlike the discrete system which will return an "all fail cases failed to match". In the fault scenario shown in

Table 2 the two most likely candidates are Sensor Fault followed by Gearbox Failure, a considerable improvement over the null answer from the discrete system.

For the purposes of integration with the Health Management architecture, the output from IFIS is fed back and needs simple post-processing in order to match the interface. IFIS output is converted in order to generate a vector of binary indicators representing faults with a very high match score and a vector for the absolute confidences in the fault isolation process.

Fault Case Description	Discrete Result	IFIS Result
FOD ingestion	Fail	58%
Sensor Fault	Fail	90%
Harness Fault	Fail	26%
Combustor fault	Fail	49%
Gearbox	Fail	70%

Table 3. Discrete vs. IFIS results for arbitrary fault scenario

Technically speaking, IFIS is integrated within the architecture as an S-Function. Since the execution time for IFIS is relatively long, the algorithm is not executed with the same frequency as the simulation, but at a lower frequency. In fact, while the architecture uses a fixed time step of 1 *ms*, IFIS is run every 250 *ms*. This low frequency is still acceptable since the system is not designed to provide a fast response to the occurrence of critical faults, but instead to provide a way to identify less serious faults which can be addressed in a longer timescale. Typically, the Health Management architecture will perform fault isolation in a timescale of 250 *ms*, then in the next 250 *ms* fault mitigation plans will be generated and proposed to the platform authority. The platform authority is expected to require an even longer time to make a decision on what plan should be applied (in the order of 10 *s*), which would render pointless any attempt to increase the frequency for fault detection/isolation or generation of fault mitigation plans.

V. Intelligent Agents integration

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 [9] 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" [10]. 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).

The reason we are interested in IAs is their flexibility. The Planning and Authorization function described in section III.C uses matrix logic and look-up tables to generate Fault Mitigation plans. With this technology, generated plans are fixed and only a certain number of parameters can be considered without excessively increasing the system's complexity. The use of Intelligent Agents (IAs) for the generation of Reversionary and Investigative Action Plans makes it possible to increase the quantity of situational awareness data which influences plan generation without exponentially increasing the complexity of the system. The reason for this is that IAs allow dynamic generation of plans, so that plans are not fixed but are instead calculated in real-time based on the current input conditions.

The IA programming tool that we selected for this project is Soar. Developed by the University of Michigan and by spin-off company Soar Technology Inc., Soar provides a robust architecture for building complex human behaviour models and intelligent systems that use large amounts of knowledge [11].

Using Soar as a development tool, a Planning and Authorization Agent was prepared. This IA performed all of the functions expected in the PA area and in fact added other functionality such as Thrust Demand Optimization. The agent is thoroughly described in another paper [12], so we would like to focus now on the process that allowed integration within the Health Management architecture.

Technically speaking, Soar is distributed as a set of libraries. These libraries must be referenced when creating the thread that executes a Soar agent. While the kernel of Soar is the same for all Soar agents, each agent must load a set of rules, which is the actual program for the agent.

Interfacing a Soar agent with the architecture can be achieved via two means: using socket communication to exchange data between Simulink and an externally running agent, or running the agent as a Simulink S-Function, which is a custom-built Simulink block, incorporating the code for the Soar kernel to effectively run an instance of Soar inside a Simulink model. To avoid synchronization problems, we chose to follow the second option.

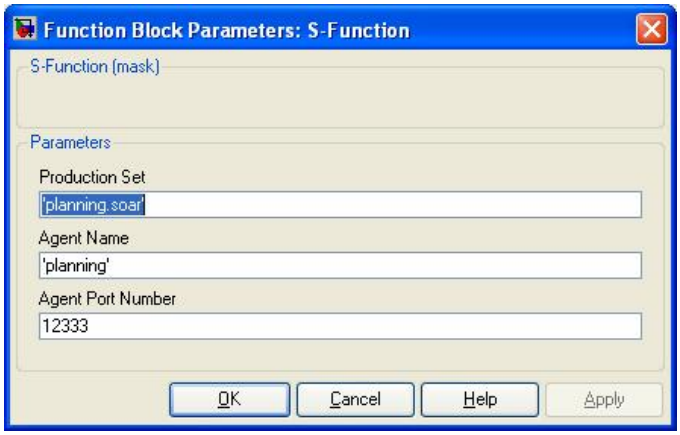

Figure 9. Soar agent interface mask

During this process we created a template for integration of Soar agents within Simulink. Since Input/Output structures are hard-coded, the S-Function must be recompiled every time they are changed. However, Soar rules are loaded from a text file at simulation start, so recompilation is not needed if changing only those. Figure 9 shows the mask used to assign the rule set and other parameters to the agent.

A testing campaign proved that not only could the Soar-based Planning Agent satisfactorily perform the same functions of the "conventional" PA sub-task, but it also allowed more capabilities to be implemented with ease. For example, Thrust Demand Optimization was easily implemented within the Soar agent, while it proved to be difficult to achieve using look-up tables and matrix logic. Furthermore, as plan generation was dynamic, a larger number of input parameters could be taken into account. In fact, plan generation takes into account not only the current fault situation, but also environmental conditions, current mission phases and other indications by the platform. Details regarding this testing campaign are out of the scope of this paper, but can be found in [12].

VI. Conclusion

In this paper, we presented the development of a complete model architecture for Gas-Turbine engine Health Management. The main objective for this architecture is integration of different types of soft computing technology. The purpose for this research was introduced and then examples of applications of soft computing to GTE Health Management and Control were presented. The architecture was then described in detail and finally two examples of integration of soft computing technology were given.

Possible future developments for this project include the consolidation of interfaces, both internal and external to the architecture, the expansion of the architecture to a multiple engine environment and an embedded real-time implementation of the architecture.

Acknowledgments

The authors would like to acknowledge the support by the DTI, the ASTRAEA Consortium and Rolls-Royce Plc in this work.

References

[1]Negnevitsky, M., *Artificial Intelligence: a guide to Intelligent Systems,* 2nd ed., Addison-Wesley, 2005, Preface.

[2]Li Y. G., "Performance-analysis-based gas turbine diagnostics: an overview", *Proceedings of the Institution of Mechanical Engineers Conference 2002 (IMechE 02),A03102, Vol 216 Part A.*

[3]Mast T., Reed A., Yurkovich S., Ashby M., Adibhatla S., "Bayesian Belief Networks for Fault Identification in Aircraft Gas Turbine Engines", *Proceedings of the 1999 IEEE International Conference on Control Applications, Kohala Coast - Island of Hawai'i, Hawai'i, USA August 22-27, 1999*

[4]Zedda M, Singh R., "Gas turbine engine and sensor fault diagnosis using optimization techniques", *AIAA-99-2530, 1999.*

[5]The Mathworks, "Simulink 7: Simulation and Model-based design", Online Datasheet, URL: https://tagteamdbserver.mathworks.com/ttserverroot/Download/43815_9320v06_Simulink7_v7.pdf (cited 11/12/2007)

[6]DePold H., Gass F., "The application of expert systems and neural networks to gas turbine diagnostics and prognostics", *Journal of engineering for Gas Turbines and Power, Vol 121, Issue 4, pages 607-612*

[7]Mills A., Tanner G., Thompson H., Fleming P., "On-wing Decision Support for Aero-Engine Line Replaceable Unit Fault Isolation", *International Symposium on Air Breathing Engines, ISABE-2007-1290*

[8]Vogel F., *Probleme und Verfahren der Numerischen Klassifikation Goettingen*, Germany, Vandenhoeck & Ruprecht, 1975, pages 28-129.

[9]Jennings N., Wooldridge M., "Applications of Intelligent Agents", in *"Agent Technology: Foundation, Applications and Markets"*, Springer, 1998

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

[11]Soar Technology Inc, "Soar – An overview", © 2002

[12]Gunetti P., Thompson H., "A Soar-based Planning agent for Gas-Turbine Engine Control and Health Management", submitted for 17th IFAC World Congress, Seoul, July 2008 (to be published)