A Maturation Environment to Develop and Manage Health Monitoring Algorithms

Jérôme Lacaille[1]

[1] *Snecma expert in algorithms, PhD, Rond-Point René Ravaud, Réau, 77550 Moissy-Cramayel cedex, France*
jerome.lacaille@snecma.fr

ABSTRACT

As the business model for selling jet engines evolves, it becomes useful to propose new systems that help the maintenance support and event monitoring. Automatic diagnosis techniques that are already well applied in other domains, such as manufacturing, chemistry, etc. may be useful in aerospace industry. There is not only a need to conceive new mathematical solutions but also to be able to manage them in time; to improve their efficiency as new data come. Every aircraft manufacturer, engine manufacturer or MRO feel this need today. This document will give a presentation of a solution for the management of HM (Health Monitoring) algorithms. The innovation process is stimulated by long-term research in university labs. The new ideas are converted to applications and codes that need to be installed in a generic framework for test and validation purposes. The maturation environment that we define here manages the ideas from the need definition to the validation of new incoming tools. Hence the development programs are able to know which are the pending innovations, understand their maturity levels and even look at the validation results. The marketing people may also anticipate the evolution of each tool to prepare the market and the business model.

1. THE ENGINE MONITORING ALGORITHM

A monitoring algorithm is a complete application that gets information from the engine or a fleet of engines and other sources, transforms data into failure indicators, detects abnormalities or unusual behaviors, and identifies faulty components. Such application is generally built from the combination of elementary steps, which are described in the OSA-CBM (Open Systems Architecture for Condition-Based Maintenance) layout. The input data are acquired from many sources and the results are reused in different applications. Moreover, an algorithm must be validated for many contexts (type of engine, flights-regimes, etc.) and a demonstration is required to validate each application.

Once selected for an engine, the algorithm is deployed on the adequate platform, which may be an embedded controller, or a ground based PLM (Product Life Management) application.

1.1 The algorithm layout

OSA-CBM framework defines 5 first layers, one for each algorithm step we used. (More layers are defined for decision management and alerts, but we will first limit ourselves with the diagnosis, prognostics and fault identification parts).

- The data acquisition layer: raw sensor observations are translated to numeric data with recording properties such as sensibility or measurement quality.
- The data manipulation layer: measurement inputs are used to compute specific fault domain indicators. At this point the engineers and experts select specific inputs relative to the monitored system. Most of the time this layer's algorithms use relationships between aircraft flight conditions and the representative input data to normalize the observations, hence making the indicators independent of the acquisition context (see figure 1).

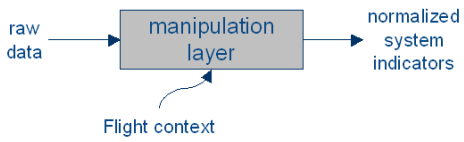

Figure 1: ***Extraction of normalized indicators.***

- The state detection layer: the observed indicators are interpreted with some physical or empirical model. This lead to the definition of a distance to the unusual behavior.
- The health assessment layer: fault indicators are converted into scores of abnormalities [R. Azencott (2003)]. One builds an anomaly probability for each known system but also a generic novelty detector for unknown behavior.
- The prognostic assessment layer: detection probabilities are accumulated and fusion with other computed results into alert anticipation. This last layer uses multiple observations confirmation method to reach the required threshold of false alarm (PFA) optimizing in the same time the probability of detection (POD).

This decomposition methodology is adaptable to any kind of monitoring. For example it may be used for inboard controller-embedded code for the vibration monitoring of the bearings but also for a ground long-term wear prognostic that uses the whole fleet of aircrafts to calibrate a supervision model.

1.2 The required measurements

The monitoring algorithms are based on observations acquired from the engine but also other sources. For context dependency analysis, aircraft information (the weight, the attitude), flight information (over land, sea…), maintenance information and manufacturing data are required.

The challenge is to build and validate algorithms; the diagnoses are calibrated on some specific data sets; some other data are used only to test the models. Moreover, data are coming from different sources: it may be operating aircrafts, but also bench tests or even results of simulations or outputs from other algorithms. Each observation (the measurements recorded during an experiment) is classified according to acquisition properties. One differentiates measurements from properties like the engine serial number, or the flight and aircraft description, etc. The properties are used as indexes to the measurements, which are stored in a relational database.

This database is capable of heterogeneous storage from events to raw high frequency measurements and maintains a high level engine-related architecture to let engineers the ability to build their own specific datasets for algorithms calibration and test.

1.3 An example list of applications

Here is a list of applications developed to improve the operational reliability. The goal of those algorithms is to limit the in-flight shutdown (IFSD), reduce delays and cancellations (D&C) but also helps the maintenance of the engine by anticipation of failures (logistic optimization) or cause component identification.

- Start sequence analysis: This algorithm analyses the behavior of the engine during the first seconds of the ignition process. It is able to detect abnormalities and classify them to target specific components [Flandrois (2009)].
- The oil consumption: a gulping phenomenon appears when the engine is running. Using the known properties of the oil and the engine configuration during the different flight phases, it is possible to compute with a good precision the total amount of oil in the engine.
- Smart filters: pressures around oil and fuel filters are used with other context measurements like temperatures and flows to anticipate the clogging.
- Bearing analysis: at any moment a buffer of vibration signals (accelerometers and tachometers) is read. Each specific failure was already modeled by a list of “flexible” known frequencies that depend on the engine regime. This algorithm spots any faulty bearing and is even able to explain the causes of the damage. Moreover, the normal behavior is also known and any new problem (another kind of damage, even a fleeting event) raises an alert and triggers the recording of high frequency data.
- Gas path analysis: snapshots of the engine thermodynamic state are taken regularly. Each data vector is normalized as to appear independent of the flight conditions and the result is scored and compared to known failure signatures [Lacaille (2009a)].
- Sensor health: using redundancy or using a correlation analysis between measurements, each sensor is tested to validate its behavior and add some data quality value (DQV) to the measurements [Lacaille (2009c)].
- Actuation loops: the stator variable geometry of the engine is controlled by an actuation loop. To analyze the temporal behavior of this loop, an autoregressive filter is modeled at any time and is compared with a reference model built during the reception test [Lacaille (2009b)].

Ground fleet analysis is also possible from the data acquired during flights and recorded in the database. A global classification map may be drawn to reference the state of each engine in a fleet [Cottrell (2009)].

2. STRUCTURE OF THE ENVIRONMENT

Our environment is based on algorithmic components (modules) that communicate using a unified protocol. The algorithm's code functions are called from the modules. A graph scheduler interfaces a standard communication scheme between modules, using a low-level DCOM/Automation layer enabling cross communications between executable processes. This way a set of tools may be connected to the toolbox of algorithmic components.

As illustrated in figure 2 three main maturation tools are now connected to the algorithms:

- The maturation interface tracks the lives of each code component in term of quality and draws maturation reports. This interface also manages development and validation processes and controls possible regression of each algorithm.
- The database manages all datasets used for calibration and validation. The database comes with a data access object (DAO) to mask the communication with the algorithms using a simple expert layer. Hence the database structure may be easily replaced without algorithm change. This feature is used to deploy application on operational ground monitoring software.
- The demonstration tool materializes each algorithmic module with a graphic representation. Using this tool engineers build new applications for new contexts just with some graphic manipulations.

To complete this environment, a standard solution for code version control manages the codes and ensures proper sharing between users. Finally a specific compiler transfers code functions on target controllers (ACMS, FADEC, etc.). Even if compilation and source control are common tools the fact that algorithms are formalized with a standard interface helps to automate the compilation and the saving process.

Once the algorithmic module communication protocol is defined, it becomes possible to replace any tool with another one (developed with a different methodology). It is also easy to insert a new tool to improve the whole platform.

3. STANDARDIZATION

Generic algorithms are programs that compute numerical results depending on input observations. The results as well as the inputs and even some sort of computation are subject to random variations, inaccuracies, and acquisition

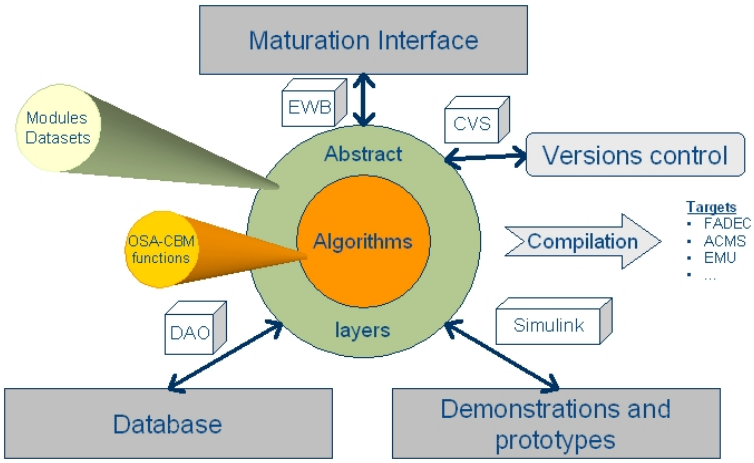

Figure 2: ***The global synoptic of the maturation environment.***

faults. A clean methodology is required to deal with those imperfections when building algorithms, calibration processes and validation plans. Even when each specific task has its own aim it is possible to formalize a unified interface to chain and schedule tasks. We need such interface to let the code stay easy to use even after a long period of time. Another requirement is the automatic control of stability to environment changes [Lacaille (2004)].

Three main objects are used in our model: one for the data materialization (the signal), one for the code encapsulation (the module) and another to represent the flow of signals through each module (the dataset).

3.1 The signal

Algorithms transform input data into output data. The inputs or outputs for each computation are contained in objects called signals. The signal has a main value, which is a set of measurements, but it also contains some properties to interpret the numeric values.

Technically, the value of a signal is contained in an array where each raw corresponds to a specific time or observation index (date, flight, engine, etc.) and each column is a measurement.

Common properties of signals are the signal name, the operation (where it comes from), the specific names of each measurement columns, the corresponding units (important when data are coming from heterogeneous sources), etc. Less common properties are the precision, the domain

bounds, etc. Some are really important for the maturation process, like quality information assign to the values. This quality is coming either from the acquisition accuracy, or from a computation chain of the efficiency of any intermediate result.

Signal objects have also specific methods for indexation, plotting purpose, and even interpolation and resampling.

3.2 The dataset

An algorithm component, a module, is working on successive observations. For each observation it gets a list of signals as inputs and produces another list of signals as outputs.

The observation corresponds to the result of a specific experiment: bench test, flight, a snapshot during a flight, engine of a given category in the fleet, etc. In general the observations belong to a set of similar objects: aircrafts owned by an airline company, flights, time measurements during a flight…

The frequency at which the output observations are produced may not be the same as the input observations frequency. Some modules produce data while others are waiting for events. Some others are resampling and use a windowing process that completely changes the scheduling of the chained modules.

The dataset object is an object that formalizes the flow of such signal lists. It manages the direct or iterative access to each observation and offers some dynamic plot methods.

Our environment will map each set of data through a named dataset object. Datasets are serialized in the database described hereafter.

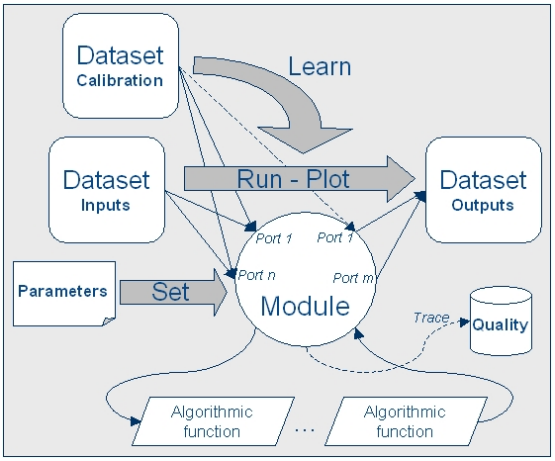

Figure 3: ***The module is a state machine that calls algorithmic functions.***

3.3 The module

Each algorithmic function is encapsulated in a module object. For an application, at least one module exists per OSA-CBM layer, but there exists also other kind of "small" modules used for signal manipulation and graphic purposes. The module object acts like a state machine. It interfaces the calls to each task used by a generic algorithmic component: state initialization, parameter change, optional calibration, transfer operation, plot, etc.

The module mainly does not contain any algorithmic code, it calls specific functions developed in toolboxes and interfaces the parameters, the resources and the state memory

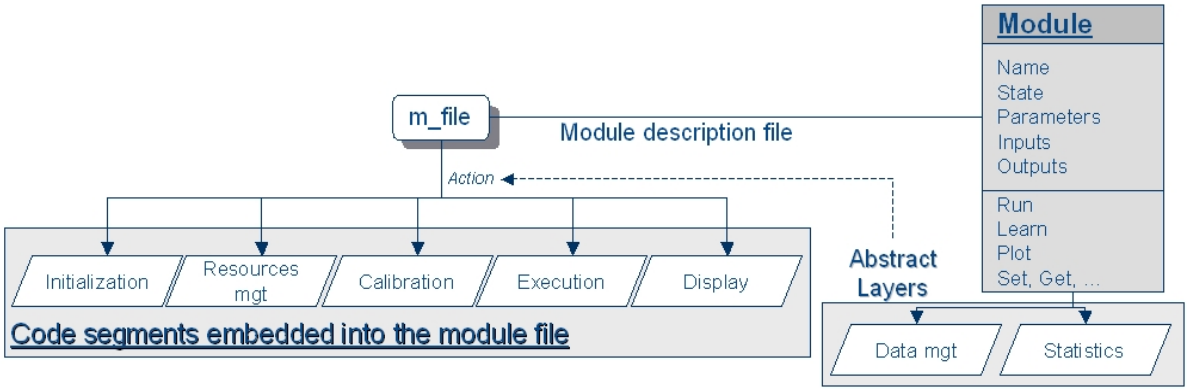

Figure 4: ***The module is an object that embeds algorithmic functions called on each specific task.***

needed to accomplish its main task (see figure 3).

But all modules must implement a specific plot function used for demonstration purpose. The plot task opens a window containing an interactive demonstration of the module computation.

For maturation purposes the module keeps track of the quality of its own treatment. A list of quality indicators are computed:

- An adequacy indicator (AQV) compares a new observation to the set of observations used during calibration and/or validation and gives a distance to the "known" distribution of the data.
- Another quality indicator is the robustness of a model (MQV) that may be obtained empirically by cross validation on test datasets.
- The genericity (GNC) is a measure of reusability of a module. It is computed as the number of applications using the module and the genericity of the functions used by the module itself.
- The TRL level (Technology Readiness Level) for a module component is obtained from the TRL of the applications that use it (the readiness level of the application for the company).

The module "TRL" is only a statistic (eg. the average) of the applications levels that use this module. Hence building a new application with only high-TRL modules is a guarantee of quality. The genericity presents another kind of quality information related to the code use.

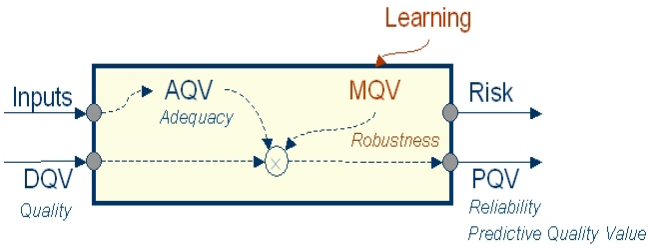

Figure 5: ***Module quality.***

With those quality indicators (see figure 5) and using the quality of the input data (DQV), it is possible for a module to define a quality indicator for the proposed outputs (PQV). This new information is stored and analyzed across time to detect trends of the module behavior (section 6).

To build a new module the user needs only to reuse the template of a specific description file that already implements standard algorithmic behaviors and instantiates his module. The module scheduler use the description file (figure 4) to ordinate each task corresponding to events and module state.

4. DATABASE

The database stores all the data required for algorithm's calibration and validation. The maturation environment ensures the permanence of

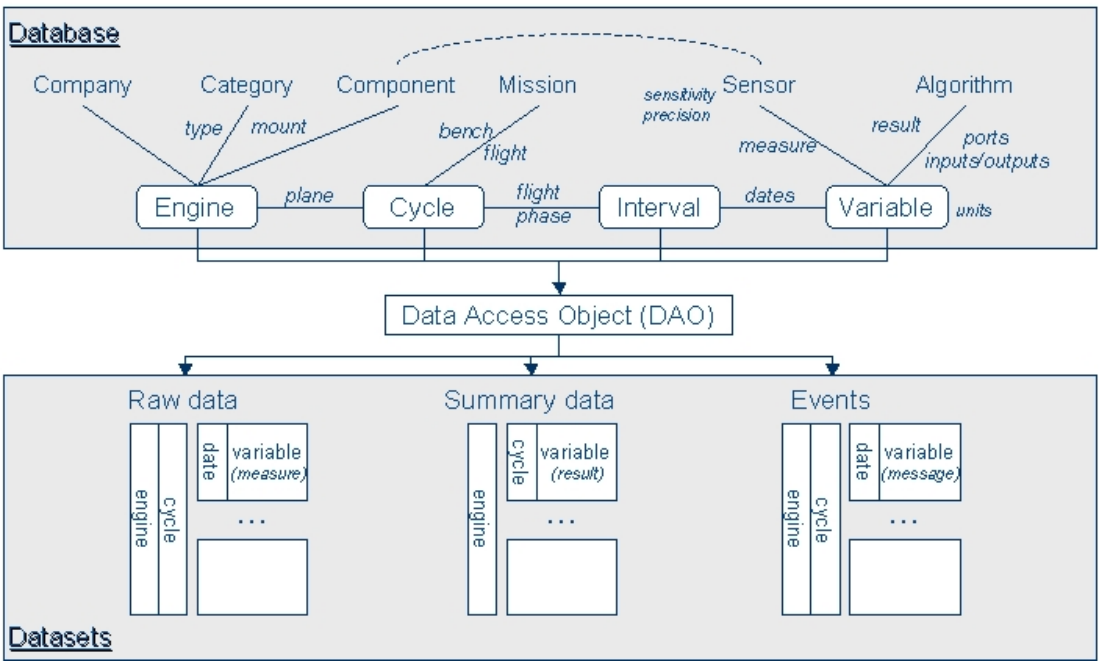

Figure 6: ***Definition of datasets from business object properties.***

each module version and each data used for calibration and validation over a very long period of time (in adequacy with the engine's life).

The algorithms are using standard signal objects as inputs (and outputs) but the signal measurements comes from very different kind of records:

- Operational aircraft measurements.
- Vibration data.
- ACARS messages sent to the ground via satellite communication.
- Engine bench test measurements from other dedicated data repositories.
- Component data and configurations from the manufacturers.
- Maintenance reports.
- Etc.

But in fact all those sources produce only three sorts of data: the events, the summary reports and temporal raw measurements. All other data may be interpreted as object's properties (engine, aircraft, component, factory, airline…).

- The events are measurements taken at random dates and are generally associated with a message and a value.
- The summaries are snapshot information taken regularly but at a small rate (for each flight and specific engine regime, each engine…). In general we associate summary data as result of statistical computation done on raw measurements.
- The temporal measurements are raw signals that are obtained from sensors or high volume outputs computed online, like spectrum.

Measurements are defined by variables and indexes:

- A variable is a data container linked to sensors or algorithms' results, which carries their own properties like bounds, sensitivity, precision.
- The index corresponds to the temporal or special coordinate that refers to the value of a variable. It may be engine-cycles (flights) or temporal intervals.

A dataset is made of simple rectangles extracted from this relational database. Each rectangle is a signal, whose columns are variables indexed by cycle or time (see figure 6).

- ACARS messages are information taken during the flight. They summarize the engines states at specific flight regimes. Classical datasets are built from those snapshots: for each engine (observation) they collect signals indexed by cycle, which record temperatures, pressures, flows, etc.
- Vibration records are taken regularly during the flight. For example at maximum acceleration or engine cut-off. Those measurements give, for each engine, datasets of cycles (observations) that collects time-indexed signals of accelerometers and tachometers.

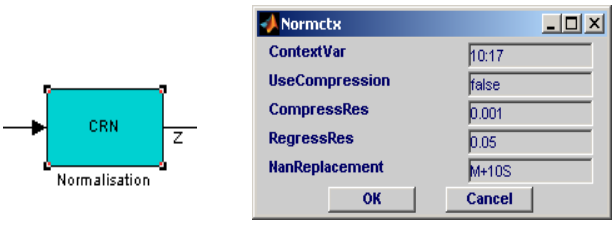

Figure 7: ***A bloc that contains an algorithmic module with its parameters.***

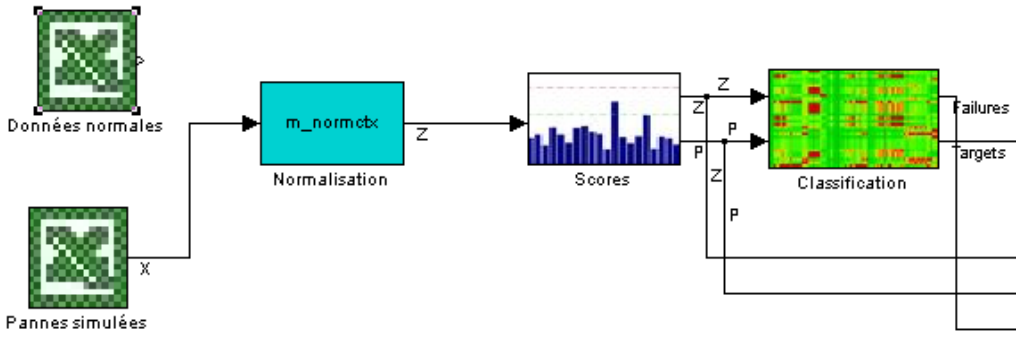

Figure 8: ***An application built from several modules.***

5. DEMONSTRATION TOOL

One of the main maturity indicators is the number of time an algorithmic component (here, a module) is reused in different applications and for different contexts (types of engines, category of missions). A way to facilitate this reuse is to present the module in a simple way. The demonstrator materializes each module by a graphic block with a simple parameterization and connection interface (figure 7).

The graph scheduler must implement a specific batch mode (for generic signal transfer across the connections) able to deal with auto-adaptability and calibration. Thus the user only needs to draw a graph of OSA-CBM modules to build a new HM application. The configuration process resembles the way the engineers are working to conceive control algorithms (figure 8).

Once a graphic block linked to an algorithmic module, it is possible to build as many instances of the module and even share the same instance by different physical block representations. Each block may execute the module in three modes: normal execution, calibration and plot. The sharing of the same instance allows the definition of a specific graph for calibration and the presence of two blocks for the same instance in the application graph: one for the computation and the other for the interactive display.

When a graph of blocks is drawn on the main working screen, it may be scheduled for calibration or execution. The specific display blocks will automatically update their graphic interfaces offering interactive demonstrations for each observation.

Executing a graph is not the only way to manage algorithms. Each block has a contextual menu (figure 9) that offers a list of actions like dynamic plot, display of inputs, outputs, re-execution, etc.

A first toolbox contains generic blocks: reader, writer, a common “module execution block”, and the “small” signal manipulations blocks.

Another toolbox of already prepared module instances is organized according to each specific application context (oil consumption, start sequence monitoring, modular performance, bearing damages, fleeting events…). This application toolbox has also a direct access to predefined datasets. Hence to get new data for some test a user needs only to drag and drop a database connected block on the screen.

The use of a share database located on a server and the automatic interface to the source control tool facilitates the sharing of data and modules between users.

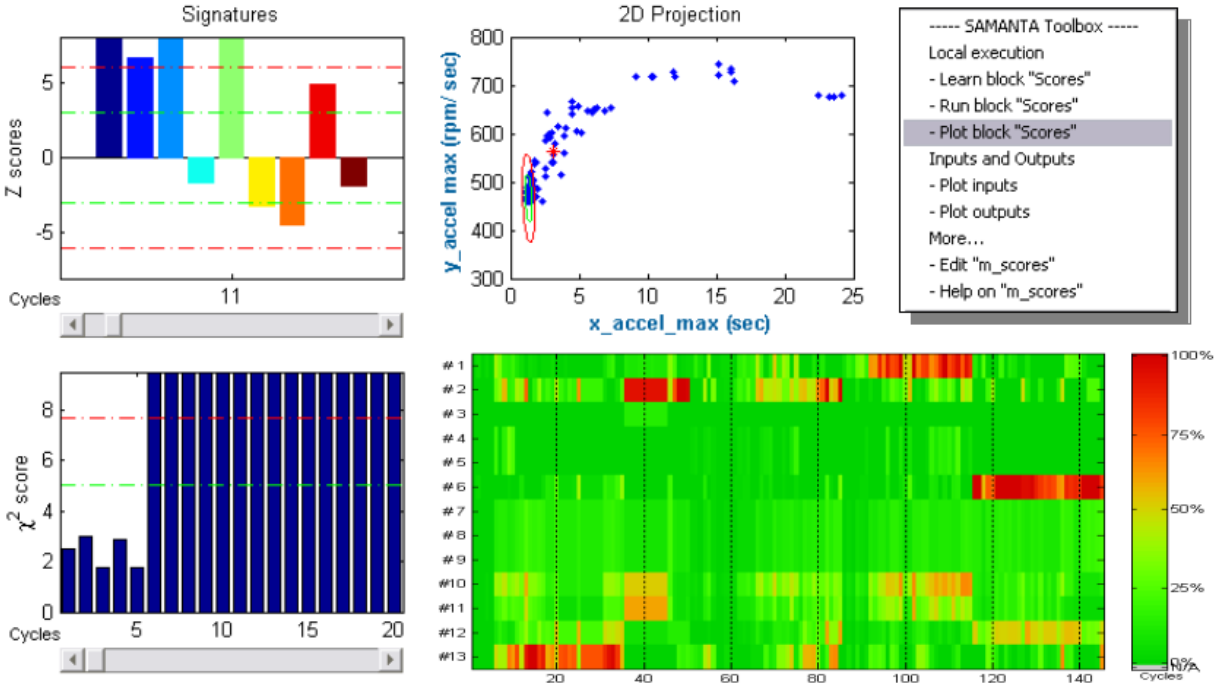

Figure 9: ***An example of interactive display interface of a module and its context menu.***

6. MATURATION INTERFACE

The main challenge for engine HM algorithm is to be able to validate the algorithms and to update their codes according to new observations. In fact, acquisition of new data, specific to a system to monitor, is certainly the major difficulty. A standard solution is to define a first version of the algorithm that is embedded for bench tests or even on real aircrafts owned by a friend airline company. This first prototype executes the current version of the algorithm but also register new observations. On operational flight we already have a communication protocol with ACMS tools for regular transfer of data to the maturation database. On ground monitoring applications we define loaders connecting the operational services to the database. Once new observations stored in the database they are used for an update calibration phase. The validation process of HM algorithm is clearly an iterative process.

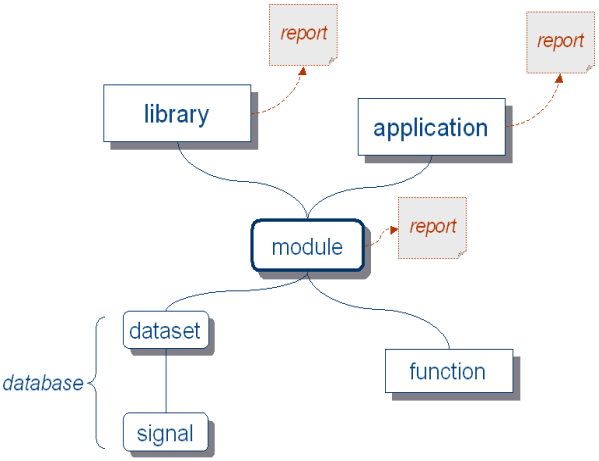

Figure 10: ***Hierarchy of objects in the environment.***

Our maturation interface is built specifically to manage this iterative process. It is called MPI for maturation process interface (figure 11). It organizes the hierarchy of components: datasets, functions, modules, libraries and applications (see figure 10 and the object hierarchy in the nomenclature section). At the base level are the algorithmic functions and the datasets. Those components are linked to modules and each module belongs to one OSA-CBM library and some high

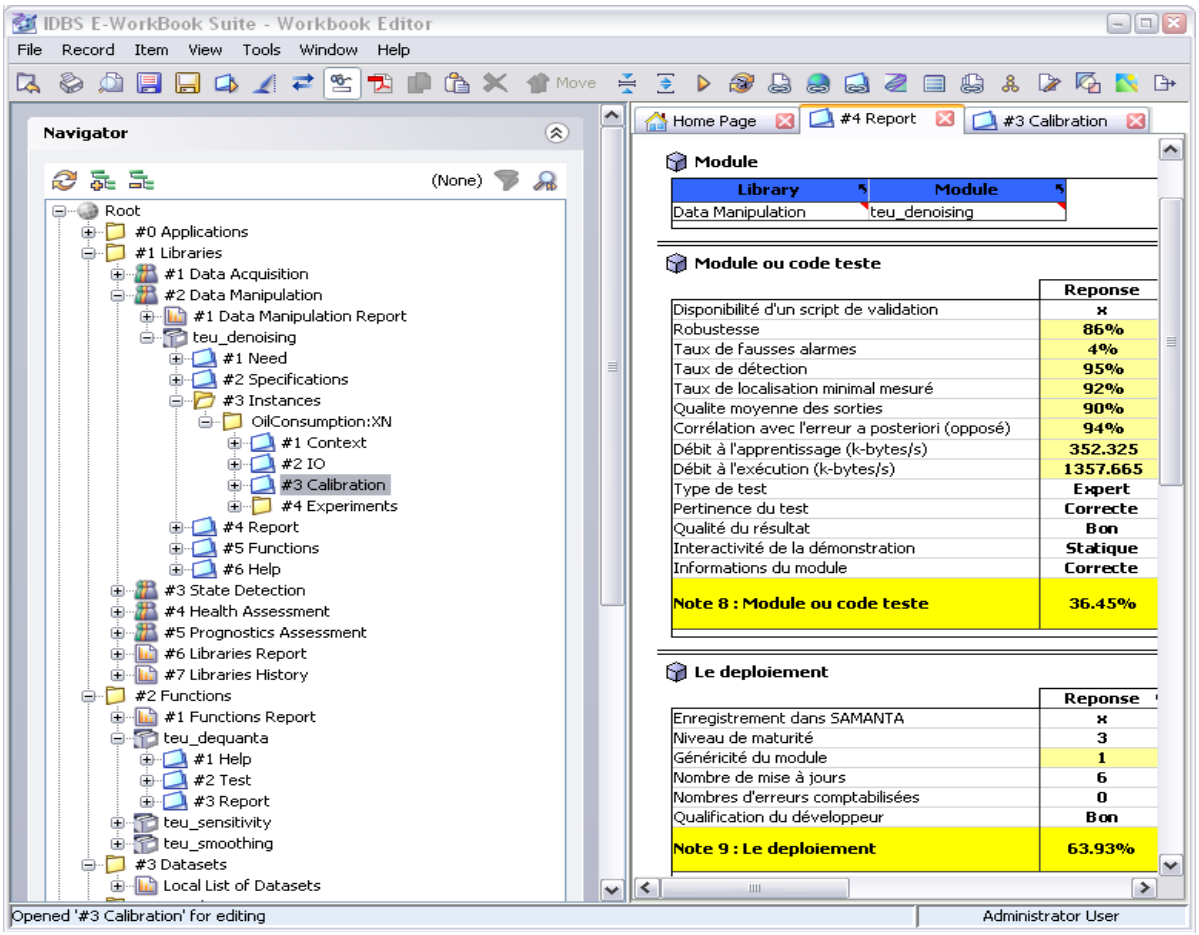

Figure 11: ***Maturation Process Interface.***

level HM applications.

When a new module is registered in MPI, the code is analyzed and all new algorithmic functions are automatically registered. Then a set of pages allows the expert engineer to calibrate a new instance linked to the algorithmic graph he is working on (from the demonstration tool). Directly from the interface, he may change parameters, select datasets, execute a calibration, launch the code and execute the interactive plot. In this work, the application helps the user by providing pages linked to the specifications, the user guide and for each instance a description of the validation context. Moreover, code development formalism defines comment syntax so the online help of any code (function or module) is automatically generated from MPI.

Once an instance calibrated and tested, reports are automatically generated for the module but also for each used function and the current algorithmic application. The modules are automatically monitored and statistic information is transferred into the reports. The user needs to complete each spreadsheet with expert comments and more high-level information as a TRL value at application level.

As MPI is built on a hierarchical structure, summary reports are automatically computed for libraries and for each application that uses the module.

Each version and modification of any experiment is saved. This allows the design of specific dashboard with temporal trends as well as a description of the current state for any module, library or application.

7. CONCLUSION

This environment is a back-office tool for a cell of people dedicated to HM algorithms development. It comes with a process for conception, maturation, validation, test and support.

The reports built by MPI use high-level statistic procedures to validate and score each algorithmic application (to monitor the health-monitoring algorithms!). Those procedures apply a methodology that uses a confirmation layer to ensure the required false alarm rate (PFA) while optimizing the probability of detection (POD). As this is only efficient at application level, our system also monitors each algorithm module for each version and each application context.

This solution was already applied in other industries. It helps a team of researchers and engineers to answer algorithmic problems in a lot of different domains: automobile, chemistry, petrol, nuclear technology, and even finance. It is also used to design solutions for semiconductor micro-fabrication SPC software in AEC and APC domains. This is our attempt to transfer such methodology in the aeronautic industry.

NOMENCLATURE

ACMS	Aircraft Condition Monitoring System
AEC	Automatic Equipment Control
APC	Automatic Process Control
AQV	Adequacy Quality Value
CBM	Condition Based Maintenance
D&C	Delay and Cancellation
DAO	Data Access Object
DQV	Data Quality Value
GNC	Genericity (reusability)
HM	Health Monitoring
IFSD	In Flight Shut Down
MPI	Maturation Process Interface
MQV	Model Quality Value
MRO	Maintenance, Repair and Overhaul
OSA	Open System Architecture
PFA	Probability of False Alarm
PLM	Product Life Management
POD	Probability Of Detection
PQV	Predictive Quality Value
TRL	Technology Readiness Level

Object hierarchy:

Application	An algorithmic solution for a specific engine system or component
Context	The environment where the application is used. It can refer to a specific engine category, a flight regime, etc.
Module	An object that embeds the algorithmic codes developed in functions. The module has a specific interface so each module may be called in a unified way.
Block	A graphic box that may be linked to a module. It gives a physical materialization to the module and can be instantiated in a graph to build an application.
Graph	A list of chained modules that may be connected to data and executes the code of an application.
Signal	An object that embeds values collected from different sensors or results of different computations. Modules use lists of signals as inputs and outputs.
Dataset	A collection of lists of signals stored

	in the database and retrieved from query on properties and variable names.
Observation	A list of signals that represent the current measurements of an experiment (flight, bench test, …) or the intermediate results of computations for this same experiment.

REFERENCES

M. Young (1995), *Cantata: Visual Programming Environment for the Khoros System*, Computer Graphics, vol. 29, pp 22-24.

V. N. Vapnik (1995), *The Nature of Statistical Learning*, Springer Verlag, NY.

A. Mertins (1999), *Signal Analysis, wavelets, Filter Banks, Time-Frequency Transforms and Applications*, John Wiley & Sons.

E. Lee (2003), *Overview of the Ptolemy Project*, University of California, Berkeley, CA, http://ptolemy.eecs.berkeley.edu/.

Miriad (2003), *Miriad Technologies supervise en temps réel les processus de production*, http://www.journaldunet.com/solutions/0303/030321_miriad.shtml.

R. Azencott (2003). *A method for monitoring a system based on performance indicators*, U.S. Patent 6594618B1, Miriad Technologies.

J. Lacaille (2004). *Industrialisation d'algorithmes mathématique*, habilitation à diriger des recherches, Université Paris 1, Sorbonne, France.

J. Lacaille (2005a), H. Dubus. *Defectivity Analysis by a Swarm of Intelligent Distributed Agents*, AEC/APC 2005, Palm Spring, CA.

J. Lacaille (2005b), *Mathematical Solution to Identify the Causes of Yield Deterioration - A defectivity data based solution with an emergent computing technology*, ISMI, Austin, TX.

M. Zagrebnov, J. Lacaille (2006), *Building a Robust Model for Process Control Using Advanced Mathematical Techniques*, AEC/APC tutorial 2006, Aix en Provence, France.

J. Lacaille (2006). *Advanced Fault Detection*, AEC/APC tutorial 2006, Denver, CO.

J. Lacaille, M. Zagrebnov (2006). *A statistical approach of abnormality detection and its applications*, AEC/APC 2006, Denver, CO.

J. Lacaille (2007). *How to automatically build meaningful indicators from raw data*", AEC/APC Palm Spring, CA.

J. Lacaille, M. Zagrebnov (2007). *An Unsupervised Diagnosis for Process Tool Fault Detection: the Flexible Golden Pattern*, IEEE Transactions on Semiconductor Manufacturing, Volume 20, Issue 4, Page(s): 355 – 363.

J. Lacaille (2008), *Global Predictive Monitoring System for a Manufacturing Facility*, U.S. Patent 20080082197A1.

OSA-CBM, *Open Systems Architecture for Condition-Based Maintenance*, Mimosa, http://www.mimosa.org/downloads/45/specifications/index.aspx.

J. Lacaille (2009a). *Standardized Failure Signature for a Turbofan Engine*, in Proceedings of IEEE Aerospace Conference, Big Sky, MO.

X. Flandrois (2009), J. Lacaille, et all. *Expertise Transfer and Automatic Failure Classification for the Engine Start Capability System*, in Proceedings of AIAA Infotech, Seattle, WA.

J. Lacaille (2009b), R. N. Djiki. *Model Based Actuator Control Loop Fault Detection*, in Proceedings of Euroturbo Conference, Graz, Austria.

S. Blanchard (2009) et all. *Health Monitoring des moteurs d'avions*, les entretiens de Toulouse, France.

M. Cottrell (2009) et all. *Fault prediction in aircraft engines using Self-Organizing Maps*, in Proceedings of WSOM, Miami, FL.

A. Ausloos (2009) et all. *Estimation of monitoring indicators using regression methods; Application to turbofan start sequence*, submitted to ESREL, Prague, Poland.

J. Lacaille (2009c). *An Automatic Sensor Fault Detection and Correction Algorithm*, accepted in ATIO, Hilton Beach, SC.

Jérôme Lacaille is senior expert in algorithms for Snecma. He joined the company in 2007 with responsibility for developing a health monitoring solution for jet engines. Jérôme has a PhD from the Ecole Normale Supérieure, France in Mathematics. Jérôme has held several positions including scientific consultant and professor. He has also co-founded the Miriad Technologies Company, entered the semiconductor business taking in charge the direction of the Innovation Department for Si Automation (Montpellier - France) and PDF Solutions (San Jose - CA). He developed specific mathematic algorithms that where integrated in industrial process. Over the course of his work, Jérôme has published several papers on integrating data analysis into industry infrastructure, including neural methodologies and stochastic modeling.