Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Control Engineering

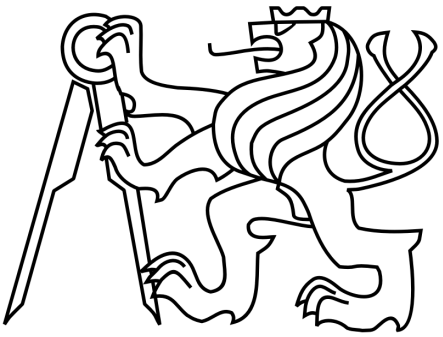
Luleå University of Technology

Department of Space Science

Kiruna Space Campus

Diploma Thesis

FAILURE DETECTION EXPERT SOFTWARE



Jana Mulačová

Erasmus Mundus programme SpaceMaster

2007

I, Jana Mulačová, hereby declare that I am the author of the following text. All the citations and references are complete and properly named.

Many thanks to my parents Jarmila and Jiří, who made me who I am, the Czech Technical University in Prague and the SpaceMaster programme that gave me education, VEGA GmbH that gave me a chance, European Union that supported me and the European culture that raised me.

Dedicated to Jens Laursen.

…….

Bc. Jana Mulačová

Czech Technical University in Prague, Faculty of Electrical Engineering

Department of Control Engineering ***Academic Year:*** 2006/2007

MASTER THESIS ASSIGNMENT

Student: Jana M u l a č o v á

Field of Study: Cybernetics and Measurement **-** SpaceMaster

Thesis Title: Failure detection expert software

T h e s i s G u i d e l i n e s :

1. Analyse approaches for fault detection and Mission Control
2. Identity and categorise types of failures with Venus Express simulator
3. Analyse approaches for failure prediction and diagnostic
4. Develope a concept for expert software.
5. Show its functionality on a prototype

Bibliography:

Supervisor: Doc. Ing. Jan Bílek, CSc.
Andreas Johansson

Assignment Date: May 2007

Thesis Due: June 2007

L.S.

prof. Ing. Michael Šebek, DrSc.
Department Head

prof. Ing. Zbyněk Škvor, CSc.
Dean

In Prague 2007-02-21

0 Contents

1 Introduction

2 About failures

2.1 FMEA/FMECA

2.2 Failure Management System

3 Problem fine definition

3.1 Example 1: Operator's error

3.2 Example 2: Loss of attitude

3.3 Example 3: Multiple failure

3.4 Operators and missions

3.5 Task identification

3.6 Task context

4 General Classification of FDI techniques

4.1 Model based filter FDI methods

4.2 Knowledge based solutions

4.2.1 Causal analysis techniques

4.2.1.1 Signed Directed Graph

4.2.1.2 Symptom tree model

4.2.2 Expert systems

4.2.2.1 Shallow-knowledge expert systems

4.2.2.2 Deep-knowledge expert systems

4.2.2.2.1 Functional reasoning

4.2.2.2.2 Causal reasoning

4.2.2.3 Shallow-deep-knowledge systems combination

4.2.2.4 Machine learning techniques

4.2.2.5 Knowledge representation

4.2.2.5.1 Rule based systems

4.2.2.5.2 Semantic network

4.2.2.6 Inference engine

4.2.2.7 Application

4.2.3 Pattern recognition

4.2.3.1 Medical application of the neural nets

4.2.3.2 Mathematical background on learnability theory

4.2.3.3 Criticism

4.2.3.4 Probably approximately correct learning

4.2.3.5 Inductive logic programming

5 Intrusion detection phenomena comparison

5.1 Anomaly detection

5.2 Misuse detection

5.3 Hybrid misuse/anomaly detection

5.4 Continuous system health monitoring

5.5 Comparison

6 Method evaluation
6.1 Applicable methods
6.2 Issues and cons of approaches
6.2.1 State-space / diagnostic model
6.2.2 Statistical methods
6.2.3 Knowledge-base
6.2.4 Neural network
6.2.5 Inductive Logic Programming
6.3 Requirements of approaches
6.4 Suggested solutions
7 Venus Express failure categorisation
7.1 FDIR System onboard
7.2 Telemetry
7.3 Tools available
7.4 VEX propulsion subsystem
7.5 VEX propulsion subsystem relevant TC/TM
8 Algorithm description
8.1 Theory
8.1.1 Introduction to expert system application
8.1.2 Knowledge base development
8.1.3 Inference engine
8.1.4 Structure
8.1.5 Logic programming extension
8.2 Practical algorithm description
8.2.1 Failures
8.2.2 Failure recognition methods
8.2.3 Subsystem specific requirements
8.2.4 Low pass filter design
9 Implementation and testing
9.1 Failure PreDetIR development
9.1.1 User requirements
9.1.2 Software requirements
9.1.3 Architectural design
9.1.4 Detailed design
9.2 Implementation
9.2.1 Modules details
9.2.2 Issues considered and encountered
9.3 Debugging and testing
10 Conclusion
11 References
Appendix A Telecommand and telemetry considered
Appendix B The Source code

1 Introduction

Failure detection, identification and recovery (FDIR) has been an important and necessary instrument for space engineering since the first space flight in 1957 and has even gained importance in 1961 with the first manned flight. As the research progressed, the satellite payload increased in value, hence becoming another strong argument for FDIR.

Nowadays, an on-board computer is commonly carried by each satellite (except of LAGEOS project perhaps), providing basic FDIR procedures in collaboration with ground operator. The ability of autonomous FDIR varies with satellite purpose, mission and equipment. However, as will be shown in part 3, even in deep-space missions with highly autonomous decisive algorithms, the on-board software (OBSW) is barely capable of handling very complex situations like multiple failures, chain reactions, externally caused failures or operator's mistakes. The autonomy of the OBSW is related to the satellite's communication possibilities. The time in view of a geostationary spacecraft is 24 hours/day, whereas the majority of the satellites merely pass by their ground station(s) several times per day with time in view in the order of minutes. Deep space missions, for a change, show the reaction time to a ground initiated command in the order of hours, a high autonomy is therefore required.

Obviously, a fully automated solution of the FDIR is needed, replacing slow reacting or far away human operator. The problem is stated as the set of failures the OBSW is incapable of preventing, detecting, identifying or recovering from.

Recently, such failures are to be handled by a human operator in the operating centre, requiring a constant attention of highly skilled and mission-specifically trained personnel, increasing expenses of the mission by the necessity of hiring and subsequent training of a team of highly educated, stress resistant specialists.

A thought occurs, an FDIR extension could be implemented within the ground control, specifically the Mission Control Systems (fig. 1.1) for all near-Earth missions, providing an open area for research of similar principle implementation into the deep space missions, to aid the operator. Such extension, in a form of a software tool, would provide the operator with situation prediction, analysis and recovery recommendations, thus improving the performance of the human operator rapidly.

Once tested and successfully operating, such tool can be tailored for and implemented to any mission, either planned or running. After proven its performance, the ground station tool might be transformed into an OBSW tool, providing especially deep space missions with higher autonomy.

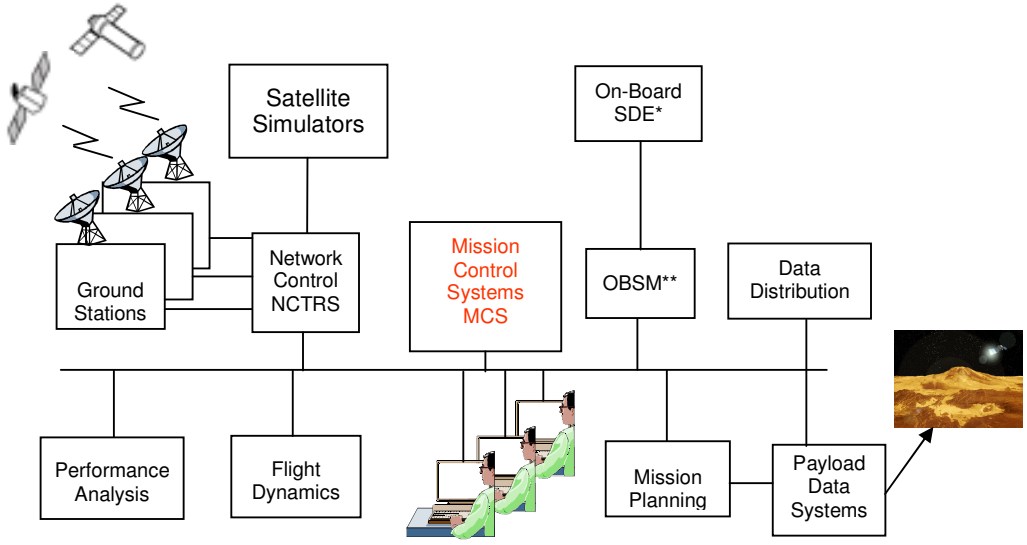


Fig. 1.1 Scope of typical ground segment [18]

*Software Development Environment

**On-Board Software Management

2 About failures

A failure is an event in the system that causes one or more unwanted performances of the system parts or behaviour. System-related classification of the failures according to Frank [1] distinguishes Instrument fault, Actuator fault and Component fault. In the satellite system, several types of failures can be defined, based on our knowledge of satellite subsystems, communication and interference with the outer space and ground and even operator human errors. However, a systematic classification and description [1] of failures is in order first.

Classification of all possible system errors distinguishes faults, modelling errors and system and measurement noise. A fault is the event, which detection is desirable. It is caused by misbehaviour of the system based on various causes from communication failure to severe hardware destruction. On the other hand, modelling errors and noise are events and influences, which are not desirable to be detected, however, can not be avoided in the system. A modelling error is an occasional difference between the examined system and its mathematical model at our disposal. The number of model errors is possible to be decreased by careful model treatment and handling of previous model errors, however, it is not avoidable. System and measurement noises are inevitable (usually electronic) noises within the system and its communication with the outer world. A noise can be caused by many factors, varying from air temperature changes in the operating centre to South Atlantic Anomaly.

Failure caused or alike changes can be abrupt (sudden, step-like) or incipient (slowly developing, bias or drift). Another issue obviously arises in recognising the incipient failures from naturally slow changes within the system.

The possibilities of system modelling are distinguished as quantitative (behavioural) and qualitative (rule-based). More about these approaches is to be found in the following chapters.

In the world of satellite failure detection, several very specific types of failures are recognised. Phantom (pseudo) failures, caused by misconfiguration of the OBSW or operator's error, are no major threat to the mission and are easy to be corrected by new OBSW upload or operator's command correction.

(Time caused) degradation of the parts, usually appearing either immediately after launch or after 10 years in orbit, is the reason why the satellite is equipped by redundant parts. The long time window in the failure appearance has been empirically observed and is related to quality of the used parts or materials. Naturally faulty part or device fails in encounter with the harsh environment of the Space, whereas the well-treated and prepared parts and devices show stable performance until the material wear-off.

Specific HW failures are the most common failures, predictable to some extent and rather easy to detect and identify within the system. Usually an OBSW FDIR is provided to handle these cases.

Unmonitored failures or their combinations are the main issue of recent FDIR research, failures the system fails to detect explicitly for various reasons, leading to the only possibility, which is the reconstruction of the situation from the output.

2.1 FMEA/FMECA

The theory and standardisation on failures (ECSS [22]), Failure Modes, Effects and Criticality Analysis (FMEA/FMECA) distinguishes between failures process, functional and hardware. Process failure is caused by a wrong utilisation, basically inevitable human error. Functional error is a failure in the functionality of the system or a device, an undesired performance. Hardware failure is a damage of a hardware part, necessarily causing subsequent functional failure.

The entire FMEA/FMECA standardisation has been developed in order to secure failure handling in industrial systems. A satellite fulfils such description, as it is an industrial system, which is rather hard to reach for manual corrections and the budget invested requires very careful error handling.

Before the system is operated, a HW/SW Interaction Analysis (HSIA) should be performed in order to avoid phantom errors or process errors caused by the software ill-behaviour.

For every part, possible failures have to be predicted and their several parameters have to be recognised. Failure severity (S) is the effect the failure causes, rated from 1 to 10, where 10 is the most severe impact on the part. Probability of the failure occurrence (P) is calculated, using specified methods, and taken into account as a factor on the scale of 1 – 10, where 10 symbolises the probability of 1. Another factor, inability to detect (D) the failure, or detectability, is rated in the inverse manner in order to achieve so called Risk Priority Number, valued between 1 and 1000 and describing the priority of the threat such failure can cause. Risk Priority Number is to be calculated using the following formula:

RPN = SxPxD

in order to provide a stable base for systematic failure classification. The unrecoverable failure of a product, designated as Single Point Failure (SPF), is naturally the utmost undesirable event.

The failure recovery on satellites naturally requires presence of redundant parts. This redundancy is distinguished as hot and cold and active and standby. Hot redundancy means the redundant part is powered in parallel to the nominally used part, whereas cold redundancy leaves the redundant part without power until the moment it is needed. Active redundancy means the redundant part is actually being used in parallel to the nominal part (for example, in TT&C system), whereas standby redundancy leaves the redundant part off until it is utilised.

The FMEA/FMECA standardisation defines certain context of a failure, including causes and consequences of each failure, and recommends the following steps:

- product definition (HW or function description)
- functional and reliability block diagrams including all items
- definition of failure modes of each item
- for each failure mode (context)
    - definition of worst consequences, severity categorisation
    - calculation of occurrence probability, probability categorisation
    - detection methods description
    - compensatory provisions
    - corrective design and actions
- documentation of analysis, critical item list development

The difference between compensation and correction in this context is following: correction is a set of actions to recover from a faulty state, whereas compensation is a design and active approach, lowering the occurrence probability of a given failure.

2.2 Failure Management System

A typical failure management system description is for example Columbus Failure Management Systems by Bade [24]. The Failure Management (FM) is being divided into three parts, Steady State FM, Reconfiguration FM and Time Critical FM.

The Steady State FM monitors the parameters and compares them to the pre-set range. In case of a failure, this FM system triggers the onboard FDIR. If the situation can not be resolved automatically, Caution and Warning (C&W) is triggered in order to await help from the station. This basic system is used when no changes are ongoing on board.

The Reconfiguration, or also called Procedural FM is on in the case the FDIR has been triggered already. This system is responsible for ensuring that each required step of recovery reconfiguration was carried on successfully. If a failure is detected, an Automated Procedure (AP) is triggered in order to start a new attempt for reconfiguration.

Time Critical (or Reflex) FM is designed to be sensitive to fast happening failures and uses the built in SW. This system does not trigger any C&W, it merely prevents any further damage. Therefore the Time Critical FM system is always on and operational and is impossible to disable or override.

The onboard FDIR equipment of Columbus project provides HW or SW based detection, identification and recovery and also crew initiated reconfigurations or repairs. The presence of human crew and the fact Columbus is part of ISS space station is where Columbus FDIR differs from satellite FDIR systems. However, the basic principles are well described on this example.

In addition to satellite-like part of the system description, another system principle is worth mentioning. As mentioned before, the ISS has the C&W system, handling alarms from various parts. However, for the Columbus cell to be self standing, it is equipped with additional system, so called EWACS (Emergency, Warning, Caution and Safing). This system collaborates with the station on C&W by safety relevant data acquisition, monitoring and processing, providing them to C&W management of the station. EWACS is a higher system than the previously mentioned 3 systems, more complex and is triggered when these three systems fail to provide sufficient reaction.

3 Problem fine definition

The on-board software of the satellite usually has the statistical FDIR methods implemented, providing prediction, detection, identification and recovery from failures caused by an obvious faulty behaviour of the physical hardware of the spacecraft. The on-ground solution is required to go beyond such performance, still keeping the ability to provide the same functionality as the OBSW.

The requirement stands for an extension to so far wide known and used methods. Extension being able to handle a situation when nothing is really wrong within the system, but the entire situation is utterly wrong due to the rest of the universe. Extension recognising the origin of a chain reaction failure and recommending switching to redundancy for the really actually damaged part, not the parts showing an error due to malfunction of the hierarchically higher parts.

In most of the satellites, OBSW provides analytical fault detection, therefore the telemetry data processed by the ground station is already pre-processed by that. However, the overall set of possible failures is not covered by situations happening within the modelled satellite.

3.1 Example 1: Operator's error

Example 1: The satellite sends an error message to the ground station and yet its state is nominal in every aspect. Explanation: the operator requested a function that is unavailable, because he forgot to switch on the device he wanted to use.

3.2 Example 2: Loss of attitude

Example 2: The satellite output is nominal, except of somewhat weak performance delivered by the solar panels is detected.
Explanation: The satellite has lost its attitude along with historical data (an utterly small part of a dead satellite hit exactly into the memory storage area, partially damaged it and gave the satellite a small drift-like torque). And the logical satellite reaction on the detected problem was switching to SAM (Sun Acquisition Mode). Using a standard procedure, the satellite found the first object bright enough to make its Sun-sensor detect stable voltage. Then, in order to recharge the battery while waiting for the recovery procedures to take action, the solar panels were turned to the bright object. The recovery procedure switched to a redundant memory storage unit, as expected and the satellite reports nominal state. The fact the satellite seems only nominal until it runs out of batteries, because the Moon doesn't provide sufficient recharge, makes such a situation time-critical.

3.3 Example 3: Multiple failure

Example 3: Multiple failures are another issue to be handled. Either a chain reaction of failures occurs or one device's failure proceeds to hierarchically subsequent parts, which detect and report an error as well. First case means the necessity to switch to the whole redundant branch and never consider any of the damaged devices useable, the other situation requires only switching to redundancy for the first, faulty part, that caused the whole series of error messages, keeping the entire branch of devices still available for future use. A decision is necessary, recognising the two dramatically different cases.

3.4 Operators and missions

The solution of above described situations nowadays is employing an operator, who has the knowledge and experience sufficient to distinguish cases from each other by observing the error message sequence, seemingly unrelated telemetry data, related historical data, hierarchically subsequent devices' behaviour, final functionality, etc.

The idea of using another model-based solution seems odd due to the complexity of knowledge, including single specific cases, which does not enable us to create any coherent model, along with the fact some model-based method is already implemented on board. However, this branch of solutions shall not be excluded.

Elementary errors, explained easily by a model-based method, are handled on board or reported to the operator as ready information. Problem comes with operator errors, OBSW errors, multiple failures and outer influences that are unpredictable and unmodellable. And that is the task for the ground station tool, to provide an operator some aid with deciphering what really happened.

Nowadays the operator is required to be experienced and to undertake a demanding training (by VEGA company, incidentally) to be aware of all possible situations to be handled. The knowledge and experience is stress-taught to future operators within several months, making them handle situations on the human-controlled simulator. (The simulator operator provides failure combinations to be solved, introduces them to the simulator and observes the capability of trainees to solve them.)

For example EUMETSAT, a beacon project of European Space. A satellite monitoring weather and providing the European meteorology institutes with meteorological images of Earth and measurement data. It is in view of the Svalbard ground station for 8 minutes every 1.2 hours. This fact causes a very stressful situation, especially considering that each specialised payload relies on another responsible expert. An expert tool is a highly desirable solution for missions comparable to EUMETSAT.

Developed tool might provide any operator with information sufficient to decide and solve the problem within less than 10 minutes, which is extremely vital, as commonly the satellite is in view of the ground station (Svalbard, for example) for such time period and stays out of view for approximately one hour afterwards.

Ground station tool is not meant to be used for deep space missions, as high autonomy is required due to the waiting period for the operator's command on transmitted error message telemetry may take 14 hours (Pluto distance) or even more.

The optimal case of mission, geostationary orbit, does not show the necessity of such tool either, as the contact with the satellite is constant. However, it is applicable for such missions in order to solve occurring problems faster.

The typical satellite, being on an arbitrary orbit around Earth, having and losing contact with the ground several times a day, is the major purpose of this work.

3.5 Task identification

The major task of the desired tool is to recognise between the reported and the real data and detect and identify whether and why the state and behaviour of the system is not as it should be. And the main requirement stands in fast operability of such tool.

The objectives of this projects are to create a generic a complex algorithm for FDIR on the ground station (specifically Mission Control Systems), which can be used and tailored to arbitrary mission.

Once finished, the differences between satellites nowadays allow the finalised tool to be re-useable, which is a desirable aspect from the commercial point of view.

3.6 Task context

The following paragraphs are supposed to clear out legal matters.

The task given in the thesis is to create a tool that might work as a satellite mission control operator hint from the FDIR point of view.

However, the whole research done and the final proof of concept are developed using the VEX (Venus Express) simulator and collaborating with VEX simulator. There is no requirement and no intention to interfere with the actual VEX Mission Control, the solution of this work will merely provide a suggestion for the future development of Mission Control Systems or their extensions. No confidential materials or data were used in this work and therefore it remains a fully open thesis, available without any restrictions.

4 General Classification of FDI techniques

As mentioned by Frank [1], advanced information processing techniques recently used are: state estimation, parameter estimation, adaptive filtering, variable threshold logic, statistical decision theory, pattern recognition and heuristic reasoning. In the following chapters, all these methods will be described and evaluated.

Today satellite onboard FDIR mainly relies on redundancy of hardware components or of software functions (e.g. associated to voting mechanisms or simple consistency checks) or on simple state vector / signal estimation techniques such as Kalman filters, and on the iterative tuning of the monitoring timers levels, quite fixed once validated with all the known delays in the signals propagation (acquisition, frequency, filtering).

A very general classification of FDI systems, not only bound to traditional approach, is given in fig. 4.1. It allows organizing the various FDI schemes depending on the design stage. Passive FDI approaches rely on constant or adaptive residual evaluation functions, in terms of threshold for the diagnosis of an occurred fault. Active FDI approaches have a residual generation branching in the type relying on hardware redundancy, in contrast to residual generation relying on analytical redundancy, produced by the means of a mathematical model of the process (incl. faults and dynamics). Hardware redundancy includes multiple copies of a same functional unit that perform the same task and can use either similar or dissimilar component to ensure robustness. The functional redundancy can be further divided into a set of approaches that rely on a mathematical model and those which are model free and with logical states, related to finite state machines. Further model free techniques are based on signal processing techniques: currently Wavelet based techniques seem to provide very good complements to the analytical based techniques.

The analytical redundancy FDI methods can be further specialized to either stochastic or deterministic versions depending on the type of the system approximation used (e.g. considered disturbance and noise profiles being either stochastic or worst case).
FDI design or the various techniques used for residual generation and evaluation, aim at optimizing a general FDI Metrics, therefore at optimizing set of specific criteria or specific metrics, attached to the best suited models representations. (e.g. discrimination of the effects causing false alarms, fault isolation, fault detection time and fault isolation time, robustness/sensitivity, tbd..)

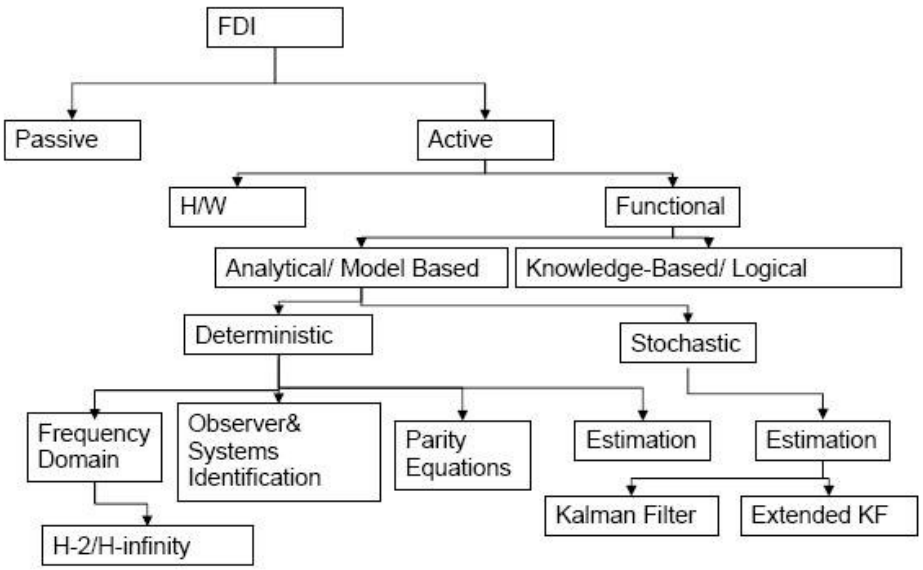


Fig. 4.1 Classification in the FDI techniques [13]

However, Chiang, Russel and Braatz [17] divide the process monitoring into three classes, data-driven, analytical and knowledge-based. Data-driven monitoring is a bottom-up approach, processing information into conclusions, opposite to knowledge-based, obtaining conclusions from given knowledge and comparing them to the data.

4.1 Model based filter FDI methods

P.M. Frank [1]: Physical redundancy is replaced by analytical (functional) redundancy – static and dynamic relationships among the system inputs and outputs

Analytical methods are of several kinds. The basic property of those methods is creating a simplified input-output model of the observed system and in certain way comparing its behaviour to the behaviour of the real system. Featuring residuals is typical for analytical redundancy methods. Residual is the result of plant observation – mathematical model confrontation and is non-zero for faulty states, disturbances, noise or model imprecision. Commonly, the fault related residual value overcomes the values of other ground's based residuals by an order. Therefore, setting the proper threshold for an error triggering is the issue of analytical redundancy.

As deduced by Frank [1], three models are needed: nominal, actual (observed) and faulty. Nominal should be updated by actual in order to reduce number of false alarms (analytical redundancy) using state or parameter estimation.

As explained by Lavagna, Sangiovanni and Da Costa [23], a proper I/O model of the system can be created by input and output sampling. Each item's input/output characteristics is to be observed with a specific sampling frequency *f* for a specific time period *t*. The sampling frequency can be derived from system dynamics knowledge and the knowledge of noise interfering with the measurement. If an unreasonably high f is required by the system dynamics, a low pass filter can be applied. The observation period has to be sufficient in order to provide two major aspects to measured data set. The properties required are completeness and minimality. Completeness condition only is fulfilled when all trajectories of the system in the phase system are included. The minimality condition is related to the curve cardinality definition, the number of points measured on each system curve has to be sufficient to uniquely describe such trajectory.

State estimation using methods are following: parity checks, observer schemes and detection filters.

A residual is defined as a difference between the models. Residuals can be used as weights for decision function or to be evaluated in a straightforward manner.

Questions: Can we by any chance model all the systems as linear SISO? Or are we supposed to have the overall system model integrated by these into MIMO?

According to Doraiswami, Diduch and Kuehner [2], FDI is implemented in 3 stages:

1) diagnostic model is developed to characterize the evolution of a feature vector as parameters in each component are subject to failure

2) residual is generated using a parity equation

3) residual is compared to a known value created from the knowledge of errors possible

Fault is detected when the residual exceeds certain threshold. The residual is linear, as long as only one failure occurs at a time. Assumption made is also that poles and zeroes of the model system are not multiple. For the application on satellite FDI, these two conditions are rather unrealistic.

The above mentioned article [2] provides full mathematical background for the solution, however, considering only an off-line solution. A significant asset of the article is the Isolability definition, which might be of a great use.

According to Frank [1], conditions for the existence of a solution are following:

- Knowledge of the nominal model
- Definitiveness of the faulty behaviour
- Existence of analytical redundancy relations
- Availability of observation reflecting the fault
- Reliability of redundant information (robustness towards unknown inputs)

Dynamical comparison between the model and the actual telemetry provides the possibility to detect out-of-limits and stay up-to-date. Such model has to include: dynamic state space model of the satellite and knowledge base for both failures and functionalities. The goal of this approach stands to find an algorithm that generates error warning under following conditions:

- The time evolution (mode) of the failure is unknown.
- The mathematical model of the nominal system is uncertain (unknown tolerances).
- There is system noise and measurement noise, which are unknown.
- The residual generation has to be done in a specified time.

Methods using analytical redundancy for residual generation are:

- **Parity space approach**: Requires accessibility of redundant measurement (model) directly. This approach is divided to direct redundancy (among redundant sensor outputs) and temporal redundancy (dynamic relation between sensor outputs and actuator inputs). Closed-loop application leads to state estimation.
- **Dedicated observer approach**: Reconstruction of the system output using measurement and observers or Kalman filters (estimation error or innovation being the residual for detection and identification) – a model feedback is the difference between the model and the real plant.
- **Fault detection filter**: (fault sensitive filter) Residual of the filter can be only unidirectional for actuator or component, not the sensor (only plane detected). Finding the matrix for model feedback is an issue of this approach.
- **Parameter identification approach**: alternative to the other three approaches, not based on state estimation. Faults of the system are reflected in the physical parameters. The idea is to identify faults by estimating the parameters of the mathematical model. This method very useful in the connection to knowledge base.

The first three approaches are obviously connected and are widely used in the OBSW FDIR.

The common, widely-used procedure is based on residuals generation followed by detection and isolation (in time, location, type, size and source).

For state estimation we can use linear or non-linear, full order or reduced order state observers in deterministic case or Kalman filters in stochastic state (noise is to be considered).

A useful depiction (fig. 4.2) [1], providing more understanding of the FDI process: I/O transfer function involves actuators, which are usually non-linear, therefore actuators are separated from the simulated model in order to maintain near-linear representation for the model.

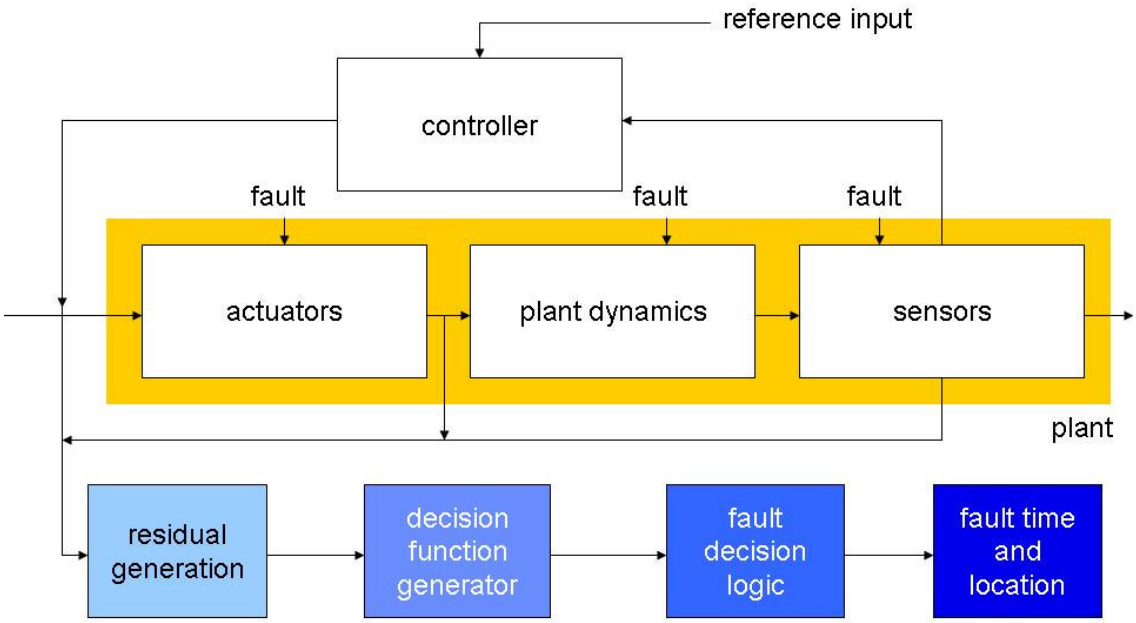
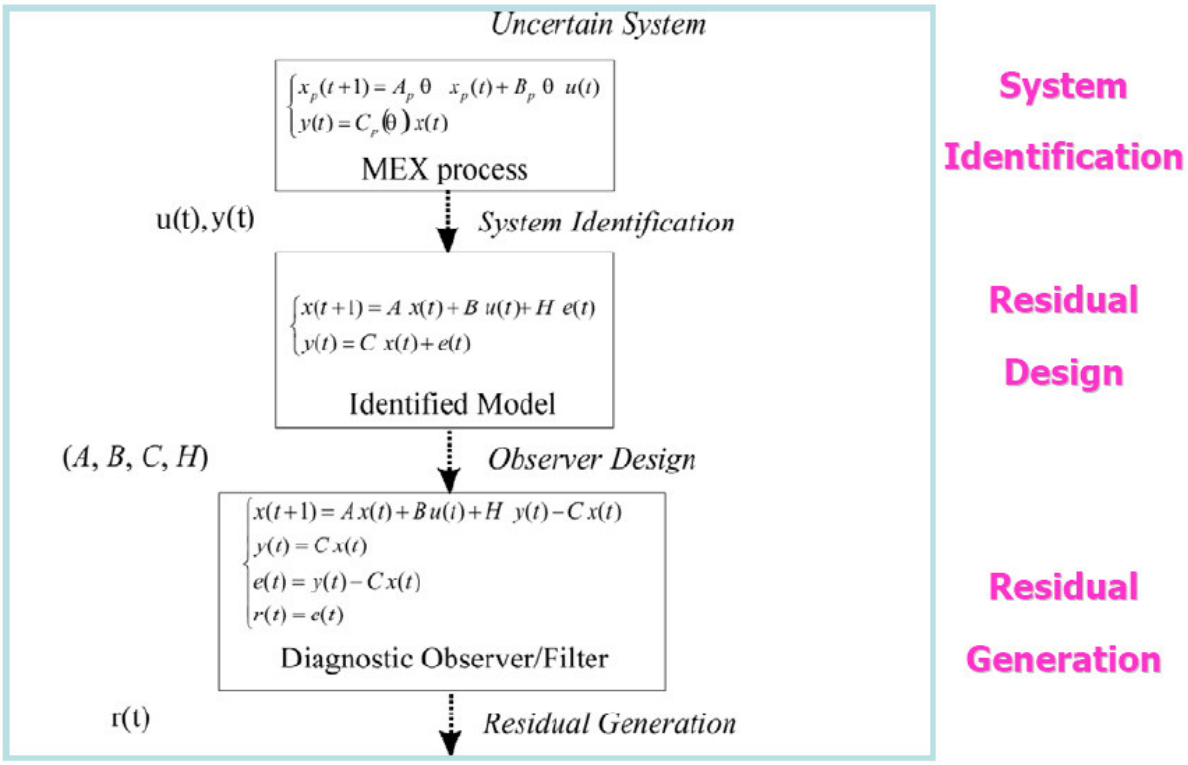


Fig. 4.2 Conceptual structure of FDI using analytical redundancy

Robustness of model based solutions is conditioned by lowering the performance, therefore a trade-off between high false alarm rate and low error detectability is necessary. The goal of FDI filter design is to generate a residual which is insensitive to external disturbances, internal system noise and uncertainties of the system model, but sensitive to fault signals. The problem of distinguishing model uncertainty symptoms from fault signals leads to trade-off between robustness and sensitivity of such filter.

After a system is modelled and the filter is designed, the filter analysis improves the filter on-line and filter validation gives us information about filter performance offline.



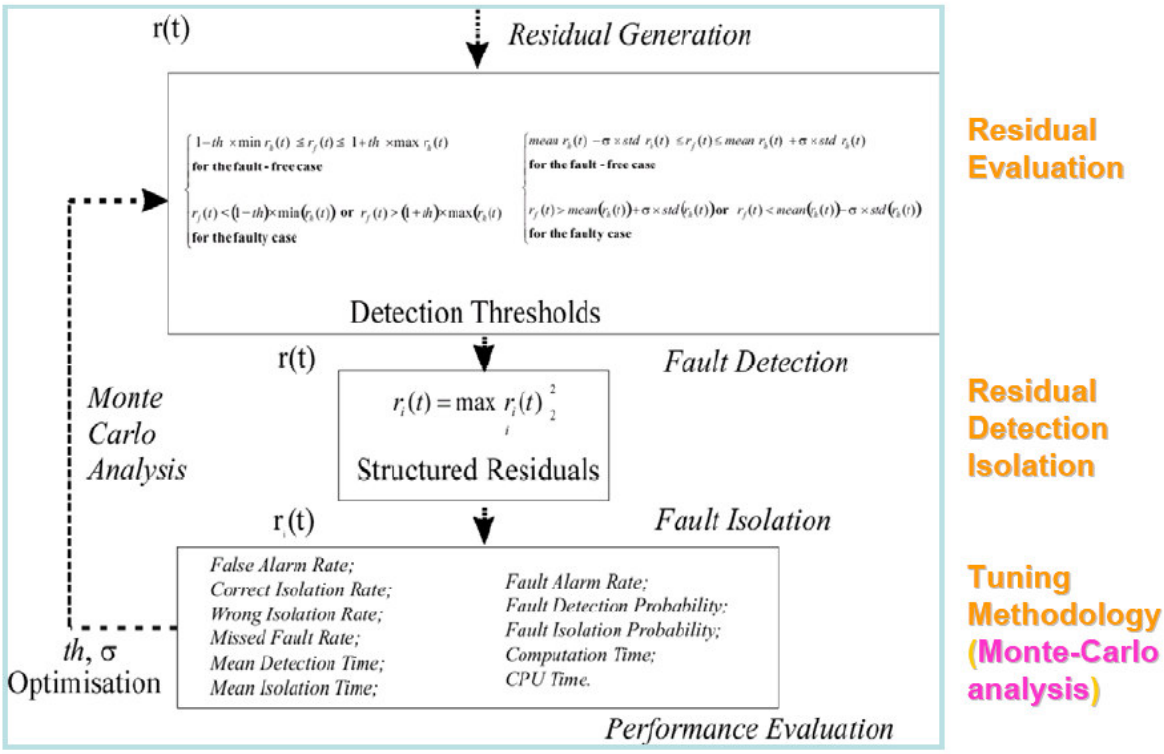


Fig 4.3 Steps usually performed for Residual Generation and Evaluation from system identification [13]

However, as proven by Johansson, Bask and Norlander [25], the model-based methods do not have to lead to a statistical treatment. The residual evaluation can be processed by a non-statistical manner.

4.2 Knowledge based solutions

According to Frank [1], unlike analytical solutions being designated a quantitative approach, knowledge base is recognised as qualitative, an approach using available knowledge of the system.

Mattos [16] states that knowledge-base solutions (meaning expert systems) are considered part of AI (Artificial Intelligence) phenomena. However, as Chiang, Russel and Braatz [17] claim, the problem of neural nets, according to Stergiou and Siganos [3] representing the Artificial Intelligence, is a department of the knowledge base world. These two contradictive statements show how incoherent and often intuitive is the environment of related research.

In this thesis I decided to follow the subordinance given by Chiang, Russel and Braatz [17].

4.2.1 Causal analysis techniques

Causal analysis techniques are utilised for system diagnostics. The main principle uses the causal modelling of fault-symptom relationships and leads to rather simple inner failure identification.

4.2.1.1 Signed Directed Graph

SDG is a qualitative model-based approach, similar to Petri nets in algorithmisation. When applying this method, a graph of nodes and arcs copying the system topology is created in order to represent the states, their possible (conditioned or unconditioned) changes and even weights on arcs representing the probability of the changes. The output of such graph is a list of most likely fault candidates. However, the main disadvantage of this method is a single fault assumption. Looking back to the method capabilities required, this method is insufficient in two main points, as it does not cover multiple neither external failure causes. However, this clumsy method can be compiled into a set of rules, which is a more agreeable modern engineering method.

4.2.1.2 Symptom tree model

Symptom tree model is a real-time version of an offline fault tree model. The root cause of a fault is determined by taking the intersection of causes of observed symptoms. However, even this method leads to an uncertain result, as a list of candidates is generated as the output. A version taking probabilities of symptom-fault pairs into account as weights on the connections. A pattern matching algorithm is a usual companion of such version.

4.2.2 Expert systems

Expert systems (often referred to as the only knowledge-based solution) imitate the human expert reasoning rules in order to reach comparable performance. Historically, the first implementations were concentrated on medical diagnostic systems. Effort has been made to expand the principle usage to other fields.

The right implementation of such system interprets the existing knowledge, accommodate existing databases, collect new knowledge, process logical inferences and provide reasoning decision. We distinguish between deep knowledge and shallow knowledge expert systems. Shallow knowledge is a rather uninformed system, deciding upon heuristics and expert delivered rules. Deep knowledge systems are provided as precise model of the problem as possible, a model either mathematical, behavioural or structural.

As mentioned by Frank [1], knowledge-base consists of:

- Sets of facts and rules
- Database of the present state of the process
- Inference engine (algorithm)
- The explanation component (to inform user)

The inference engine combines the analytical approach with the knowledge, it has to have an access to the analytical model structure and parameters, heuristic knowledge of fault propagation, statistics operational and environmental conditions, process history, etc. and actual data (input, output, operating conditions, …)

4.2.2.1 Shallow-knowledge expert systems

Also known as experiential knowledge or empirical reasoning systems. The expert experience and knowledge is formulated into a set of IF-THEN rules, which are used to achieve diagnostic deduction. The reasoning of such system is flexible and transparent, as the knowledge is presented as a data processing rule from the very beginning. The efficiency and

results of such system depends strongly of the quality of the given rule set, the adequacy and well-formulation of the knowledge provided.

Shallow-knowledge expert systems are not intended to produce solutions that have not been delivered previously. Therefore the proper knowledge acquisition becomes the crucial step in a construction of such system. Especially for a large scale system, the proper knowledge base development is a time consuming, demanding matter.

However, once constructed, such expert system provides the same performance as a human expert, acting based upon experience.

4.2.2.2 Deep-knowledge expert systems

Also known as model-based, functional reasoning or diagnosis-from-first-principles. A deep knowledge expert system is based on engineering fundamentals of the given plant, like structural description and behavioural rules of its components in both faulty and nominal state.

Deep knowledge systems are utilised when a complex, novel or explanation requiring situation occurs. Reasoning on causal and functional information is involved in the data processing.

4.2.2.2.1 Functional reasoning

Not very different from analytical methods, functional reasoning uses the knowledge of the principles, which govern the observed process, processed into a set of equations using physical laws. Such equations determine constraints for the process variables, which can then be observed and compared. Each constraint violation has a known set of causes within the observed system.

4.2.2.2.2 Causal reasoning

This method requires definition of one rule for each possible fault origin. Such rules are combined in order to deduce all failure suspects.

4.2.2.3 Shallow-deep-knowledge systems combination

Practical implementation of the knowledge-based systems shows that the most efficient solution is the combination of the deep and shallow knowledge. A complex system is then developed, using information on system documentation, functionalities of particular components, system interrelationships and device failure history and heuristics.

Developing a first-principles model of a large-scale system is costly and demanding. Therefore the deep knowledge can be converted into production rules in order to support the shallow-knowledge-based system's performance.

4.2.2.4 Machine learning techniques

The main issue in knowledge based solutions is the knowledge acquisition and interpretation. Formulating life-long experience into simple rules can be a very demanding task. Neural

network development allows automatisation of the process, recognising patterns in data in order to figure out the rules. Fuzzy rules can be employed.

4.2.2.5 Knowledge representation

4.2.2.5.1 Rule based systems

The simplest representation of the expert information is a set of IF-THEN rules. Therefore a common design of the expert system is based on the following parts: rule base, working memory, rule interpreter.

The rule base consists of rule clusters, each responsible for encoding the knowledge required for a certain task.

Working memory is the actual database including data, inferred hypotheses and internal information about the program.

Rule interpreter is a mechanism designed to select and evaluate rules.

The rule-based system design requires a homogeneous knowledge representation and allows knowledge growth through new rules implementation.

4.2.2.5.2 Semantic network

A semantic network is a method of knowledge representation in which concepts and relations are represented as nodes and arcs, respectively. Additional program, able to maintain the relationship between the network and the meaning it represents, is necessary.

Good implementation of such program is a system of frames, collections of related nodes, providing a description of an object or event. Hierarchical structure of such frames provides relationships between domain objects in order to complement set of rules as a description of the objects which comprise the domain.

4.2.2.6 Inference engine

The inference engine mechanism gathers the needed information in order to draw inferences or conclusions for the process involved and presents obtained inferences or conclusions with explanation or bases.

There are several approaches of the inference reasoning, leading to two most used solutions.

The backward/forward reasoning combines two approaches. The backward chaining searches for evidence for a hypothesis, whereas forward reasoning creates the hypothesis based on the data.

Another approach [28] is the hypothesis/test method, approximating human diagnostic reasoning. After the observation, a hypothesis is deduced and subsequently its known symptoms are checked against the data. If the hypothesis does not hold for the sufficient proof, another hypothesis is created until the whole space of hypotheses is exhausted.

4.2.2.7 Application

According to Mattos [16], Knowledge Base Management requires specification of 3 classes of functions to process, KB online construction, Knowledge use and KB maintenance (ensuring the base efficiency and integrity).

A Knowledge system should offer KR schemes that permit an appropriate representation of all types of knowledge and allow a description of knowledge which is independent of the application programs.

Operations provided by the fully implemented knowledge system functions are storing, retrieving and deriving new knowledge.

On page 60 Mattos [16] mentions the possibility to use the knowledge systems for diagnostics in medicine – that means usefulness for satellite diagnostics – and planning in robotics – that might mean predictive ability.

4.2.3 Pattern recognition

Pattern recognition provides a non-rule-based reasoning, failure detection and identification based on observation of significant patterns. All known approaches to FDI are able to incorporate pattern recognition to some extent. However, the widely used approaches, implementing pattern recognition in order to avoid modelling the internal process states or structure explicitly, are Artificial Neural Networks and Self-organising Maps. These approaches are recommended to be used for abundant data cases or expert knowledge lack.

The definition of Neural Networks according to Sarle [9] is following:

"*There is no universally accepted definition of an NN. But perhaps most people in the field would agree that an NN is a network of many simple processors ("units"), each possibly having a small amount of local memory. The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training.*
Some NNs are models of biological neural networks and some are not, but historically, much of the inspiration for the field of NNs came from the desire to produce artificial systems capable of sophisticated, perhaps "intelligent", computations similar to those that the human brain routinely performs, and thereby possibly to enhance our understanding of the human brain.
Most NNs have some sort of "training" rule whereby the weights of connections are adjusted on the basis of data. In other words, NNs "learn" from examples, as children learn to distinguish dogs from cats based on examples of dogs and cats. If trained carefully, NNs may exhibit some capability for generalization beyond the training data, that is, to produce approximately correct results for new cases that were not used for training."

4.2.3.1 Medical application of the neural nets

An interesting relation occurs between the medical utilisation of the Neural nets and the satellite FDIR purposes. Citing from [3]: "*Artificial Neural Networks (ANN) are currently a 'hot' research area in medicine and it is believed that they will receive extensive application to biomedical systems in the next few years. At the moment, the research is mostly on modelling parts of the human body and recognising diseases from various scans (e.g. cardiograms, CAT scans, ultrasonic scans, etc.). Neural networks are ideal in recognising diseases using scans since there is no need to provide a specific algorithm on how to identify the disease. Neural networks learn by example so the details of how to recognise the disease are not needed. What is needed is a set of examples that are representative of all the variations of the disease.*

The quantity of examples is not as important as the 'quantity'. The examples need to be selected very carefully if the system is to perform reliably and efficiently.

Neural Networks are used experimentally to model the human cardiovascular system. Diagnosis can be achieved by building a model of the cardiovascular system of an individual and comparing it with the real time physiological measurements taken from the patient. If this routine is carried out regularly, potential harmful medical conditions can be detected at an early stage and thus make the process of combating the disease much easier.

A model of an individual's cardiovascular system must mimic the relationship among physiological variables (i.e., heart rate, systolic and diastolic blood pressures, and breathing rate) at different physical activity levels. If a model is adapted to an individual, then it becomes a model of the physical condition of that individual. The simulator will have to be able to adapt to the features of any individual without the supervision of an expert. This calls for a neural network.

Another reason that justifies the use of ANN technology, is the ability of ANNs to provide sensor fusion which is the combining of values from several different sensors. Sensor fusion enables the ANNs to learn complex relationships among the individual sensor values, which would otherwise be lost if the values were individually analysed. In medical modelling and diagnosis, this implies that even though each sensor in a set may be sensitive only to a specific physiological variable, ANNs are capable of detecting complex medical conditions by fusing the data from the individual biomedical sensors.

ANNs are also used experimentally to implement electronic noses. Electronic noses have several potential applications in telemedicine. Telemedicine is the practice of medicine over long distances via a communication link. The electronic nose would identify odours in the remote surgical environment. These identified odours would then be electronically transmitted to another site where an door generation system would recreate them. Because the sense of smell can be an important sense to the surgeon, telesmell would enhance telepresent surgery.

Another application developed in the mid-1980s called the "instant physician" trained an autoassociative memory neural network to store a large number of medical records, each of which includes information on symptoms, diagnosis, and treatment for a particular case. After training, the net can be presented with input consisting of a set of symptoms; it will then find the full stored pattern that represents the "best" diagnosis and treatment."

4.2.3.2 Mathematical background on learnability theory

The goal of Valiant's paper [15], a beacon of neural network theoretical works, is to state conditions for learning, create a concept of learnability, comparable to computability, which answers what can be calculated. Learning machines are bound to the topic of human experience interpretation. Human being behaviour consists of genetically given reactions, memorised actions and remaining large area of skill acquisition, which can be called learning. The learning can be simplified to an answer to whether a concept Q is true or not for given data. We say that concept Q has been learned if a program for recognising has been developed.

Several terms have been defined to support this theory:

Learning machine description: It can provably learn whole classes of concepts, which can be characterised. The classes are appropriate and non-trivial for general-purpose knowledge. The resulting computational process for the machine to deduce the required program is feasible (has feasible or polynomial number of steps.

Learning protocol: manner of obtaining the information from the outside

Deduction procedure: mechanism of the final program development

The network learning concept consists of two parts, protocol definition and delivering a class of concepts learnable in polynomial time.

Two methods for neural net learning are known, a set of **examples** fulfilling the concept or **oracle**, set of random examples with information whether the concept is fulfilled by them.

Deduction procedure results in an expression that approximates the real learned algorithm as close as possible. The result of such expression is yes when it shouldn't be no, but can be no when positive answer is appropriate. The false no rate can be arbitrarily diminuated by introducing more learning steps. This is called one sided error learning.

Neither examples nor oracle's answers should be filling the whole space of possible, only describing the naturally possible.

However, a well defined, easy to describe and compute function might be not computable without knowing a key (cryptography) and therefore not learnable by example or oracle. Therefore if we know an algorithm, we preferably not use a learning algorithm to discover it again. A program is learnable if and only if there is an algorithm A invoking the protocol with properties:

According to Haykin [21], time of algorithm is polynomial in an adjustable parameter h. Probability of $(1-h^{-1})$ of creating a program that never gives 1 when it shouldn't and almost each time gives 1 when it should. This definition leads to Probably Approximately Correct learning, part 4.2.3.4.

4.2.3.3 Criticism

Citing from Wikipedia [4]:

"A. K. Dewdney, a former Scientific American columnist, wrote in 1997, "*Although neural nets do solve a few toy problems, their powers of computation are so limited that I am surprised anyone takes them seriously as a general problem-solving tool*." (Dewdney, p.82)

Arguments against Dewdney's position are that neural nets have been successfully used to solve many complex and diverse tasks, ranging from autonomously flying aircraft to detecting credit card fraud.

Technology writer Roger Bridgman commented on Dewdney's statements about neural nets: "*Neural networks, for instance, are in the dock not only because they have been hyped to high heaven, (what hasn't?) but also because you could create a successful net without understanding how it worked: the bunch of numbers that captures its behaviour would in all probability be "an opaque, unreadable table...valueless as a scientific resource"*.

In spite of his emphatic declaration that science is not technology, Dewdney seems here to pillory neural nets as bad science when most of those devising them are just trying to be good engineers. An unreadable table that a useful machine could read would still be well worth having."

According to Karel Macek, employee of Advisory Services, PricewaterhouseCoopers, Czech Republic, a neural network specialist, there are important aspects of the use of neural networks that are necessary to be mentioned:

"*1) Neural network simulates the process of learning. For its use, usually some (large) set of data is necessary to provide "study materials".*

2) Neural network is suitable for solving problems uncertain, complicated (complex) or resource demanding in the direct approach. A neural network could be used for 1+1 calculation, but is not due to the higher system costs and certain uncertainty of such solution.

3) There are various architectures of NN, which can be used for various purposes.

The power of NN is mainly in the following aspects:

1) Searching for relations (function approximations): if a y is dependant on an x, most of the architectures are able to discover this relation and then "guess" a y for unknown x. Moreover, the net can, when treated with attention, generalise well, meaning it has no "unreasonable prejudice".

2) It is possible to use NN for the input dimension reduction. If it is x being of a subset of R^n and the n is large, it can cause trouble. NN can help to estimate which elements are relevant and which can be neglected.

3) NN can be used for fuzzy rules extraction. In my opinion, this contribution to the control theory is quite major. An inverse pendulum (balancing a club on a palm) can be controlled upon trivial rules, whereas solution using differential equations is extremely complex.

4) NN are commonly used in robotics, e.g. controlling a robot arm (the relation between the moves and the goal is complicated, but thanks to the net the arm learns to succeed) or when recognising a surface (control parameters)."

Regarding the satellite failure detection application, Mr. Macek believes the NN is able to learn primarily what "feeling" it has about each component. Then the technology offers the possibility to extract this feeling, express it explicitly and apply it in the final control algorithm. It is a possible alternative to ease the development part, which is very demanding, or to provide a better solution if the final algorithm is time consuming. The neural networks can simplify the whole problem in such cases. However, no capability of overall solution is provided by the neural net methods.

4.2.3.4 Probably approximately correct learning

A more scientific argument against the usage of Neural networks is simply the calculation of Probably approximately correct learning requirements.

Citing from Wikipedia [11]: "*Probably approximately correct learning (PAC learning) is a framework of learning that was proposed by Leslie Valiant in his paper A theory of the learnable.*

In this framework the learner gets samples that are classified according to a function from a certain class. The aim of the learner is to find a bounded approximation (approximately) of the function with high probability (probably). The learner must be able to learn the concept given any arbitrary approximation ratio, probability of success, or distribution of the samples.

The model was further extended to treat noise (misclassified samples). The PAC framework allowed accurate mathematical analysis of learning.

Also critical are definitions of efficiency. In particular, finding efficient classifiers (time and space requirements bounded to a polynomial of the example size) with efficient learning procedures (requiring an example count bounded to a polynomial of the concept size, modified by the approximation and likelihood bounds).

PAC learning framework is part of computational learning theory."

Citing Sarle [9]: "*Feedforward NNs are restricted to finite-dimensional input and output spaces. Recurrent NNs can in theory process arbitrarily long strings of numbers or symbols. But training recurrent NNs has posed much more serious practical difficulties than training feedforward networks. NNs are, at least today, difficult to apply successfully to problems that concern manipulation of symbols and rules, but much research is being done.*"

The following definition by Valliant [15] is of interest:

L(h, s) is defined (for R ∋ h > 0, Z^+ ∋ s) as the smallest integer such that in L independent Bernoulli trials each with probability 1/h of success, the probability of having fewer than s successes is less than 1/h. For s ≥ 1 and h > 1,

$$L(h, s) \leq 2h(s+\log h). \quad (1)$$

Citing [12]: "*A conjunctive normal form (CNF) is a product of sums, meaning an **and** of **or**s. Valiant [15] requires each clause c_j in a CNF to be a sum of literals, where a literal is either a variable p_j or a negation of a variable. For example, $p_2+p'_3+p_6$ is a clause. In a k-CNF, each clause contains at most k literals. Theorem: for each k > 0, any k-CNF is learnable via an algorithm that uses $L(h, (2t)^{k+1})$ calls of EXAMPLE and no call of ORACLE.*"

For a satellite, receiving 27000 pieces of independent information (Venus Express) in single telemetry set, for ease considering them Boolean values and only one third of them being relevant and required to be processed by the neural net, the learnability of the algorithm of 80%, k = 9000, we can provide the learning success rate of 0.5, therefore h = 2.

The value of $L(2, (2t)^{9001})$ is reached and applying (1), the following result is obtained:

$$L(h, s) \leq 4((2t)^{9001}+\log 2). \quad (2)$$

The value of t is defined as polynomial time of algorithm, which brings to the overall calculation another complication, as with increasing t the L increases, leading to increase in t, etc.
Obviously the ≤ sign shows the possibility to obtain a lower number of examples needed, however, this algorithm calculates the necessary number of examples to be provided to the neural net in order to reach the probability of learning over a given limit.
Such number of examples is rather unavailable for any mission, as the satellite lifetime is limited to several decades. The amount of examples necessary for such cardinality neural network to learn is comparable to 10^{3000} years of standard near-Earth telemetry data transmission.

4.2.3.5 Inductive logic programming

The world of automated learning and data mining is however not limited to neural approach. Alternative methods have been developed, surprisingly using the "traditional", algorithmic approach. According to Muggleton [14], Inductive Logic Programming (ILP) is a research area formed at the intersection of Machine Learning and Logic Programming. ILP systems develop predicate descriptions from examples and background knowledge. The examples, background knowledge and final descriptions are all described as logic programs. A unifying theory of Inductive Logic Programming is being built up around lattice-based concepts such as refinement, least general generalisation, inverse resolution and most specific corrections. In addition to a well established tradition of learning-in-the-limit results, some results within Valiant's [15] PAC-learning framework have been demonstrated for ILP systems. U-learnability, a new model of learnability, has also been developed.

According to Lavrač and Džeroski [20], recently successful applications areas for ILP systems include the learning of structure-activity rules for drug design, finite-element mesh analysis design rules, primary-secondary prediction of protein structure and fault diagnosis rules for satellites.

5 Intrusion detection phenomena comparison

There is a significant conceptual similarity between the phenomena of satellite failure detection and network intrusion detection. Both problems require operation in a communication environment with uninsured, yet robust protocol, both are an important protection against a huge investment loss, both operate with packets of data, processing them into information about a health status, both are to some extent using and considering similar approaches. The difference happens to be the fact that a living hacker shows more invention and interactivity than nearly empty space environment. However, as the methods used in intrusion detection are designed to handle the worse of the cases, satellite failure detection can only profit from this comparison.

Unlike workflow system exception handling system [10], for example, intrusion detection methodology provides a whole branch of experience, very useful for spacecraft FDIR purposes.

James Cannady [19] defines misuse detection as follows:
"*Misuse detection is the process of attempting to identify instances of network attacks by comparing current activity against the expected actions of an intruder.* "
However, intrusion detection is properly defined as an attempt to discover an unwanted or extraordinary activity within the usual network traffic. Nowadays threat of data manipulation, copying or even destruction has the background in the general digitalisation of the society. Hackers, network attackers, are able (when successful) to access, change or delete confidential data like bank accounts, government or army related information or even private e-mail accounts and web pages. The trust in data of nowadays society requires well operating data and network anti-intrusion protection.

Halme and Bauer [7] define the main anti-Intrusion concepts as following:

Prevention precludes or severely handicaps the likelihood of a particular intrusion's success.

Pre-emption strikes offensively against likely threat agents prior to an intrusion attempt to lessen the likelihood of a particular intrusion occurring later.

Deterrence deters the initiation or continuation of an intrusion attempt by increasing the necessary effort for an attack to succeed, increasing the risk associated with the attack, and/or devaluing the perceived gain that would come with success.

Deflection leads an intruder to believe that he has succeeded in an intrusion attempt, whereas instead he has been attracted or shunted off to where harm is minimized.

Detection discriminates intrusion attempts and intrusion preparation from normal activity and alerts the authorities.

Countermeasures actively and autonomously counter an intrusion as it is being attempted.

Obviously, prevention as defined here is not of our concern, as in the satellite failure detection parallel, such procedures can only be performed in the HW/OBSW construction stage.

Pre-emption, however, fulfils an important function, seemingly comparable to what we call prevention, composed of prediction and operator warning. Unlike intrusion pre-emption, predictive algorithm of failure detection itself does not operate in an offensive meaning. The operator, provided a warning, can start a preventive operation (switching to redundant reaction wheel when the operating wheel's temperature sensor notices increase over certain boundary) or decide not to take the warning into account, if false alarm situation is apparent.

Thus leading to what we define as prevention of satellite failure being classified as Intrusion Detection.

Deterrence and deflection are to be considered solely for the cases of the S/C encountering extra terrestrial life form or competition agency astronaut or an alien S/C trying to use the ground station fault detection for itself. As mentioned previously, very improbable cases are to be omitted.

Detection definition is exactly following the meaning of fault detection, leading to significant similarity between the two explored phenomena.

Countermeasures lead to a fully autonomous system, which is highly desirable for the OBSW, however, not necessary and due to higher risk of missed alert or counter measuring a phantom fault not eligible on ground.

Halme and Bauer [7] distinguish methods of Intrusion detection, described in the following chapter, which might lead to parallels in the world of fault detection.

5.1 Anomaly detection

Anomaly detection, also referred to as behaviour-based [6] or statistical [8] – the normal network traffic is known and anything extraordinary is reported (discovers extraordinary behaviour even if that is not defined as a known misuse)

Anomaly detection methods:
Threshold detection: observes values of system variables and triggers alarm if they reach a threshold.
Threshold detection is defined identically with the same method in satellite failure detection. However, such simple detection and identification is the matter of OBSW, not needed to be done on the ground station.
User work profiling: compares particular user's standard behaviour to the recent one in short and long term.
User behaviour can be closely related to device behaviour onboard the satellite. This is a very common method, implemented in the OBSW.
Group work profiling: assigns users to groups with similar behaviour.
The idea of grouping devices by behavioural patterns and not their particular functions brings a simplification into final solution definitions.
Resource profiling: monitoring and generalisation of the usage of system resources
Monitoring the systems resources is one of the naturally implemented OBSW tools.
Executable profiling: user-independently observes programs executed in order to reach resources.
This application specific method is inapplicable for the majority of the satellite subsystems and senseless for the remaining ones.
Static work profiling: monitoring users does not allow them to slowly broaden their activities without informing the administrator.
In the means of satellite fault detection, a stable set of boundaries is only changed when change of mission stage or required functionality is upcoming.
Adaptive work profiling: pre-filters incoming events into three categories, passing the filter and broadening the furthermore filtering, passing the filter and not being used to broaden the statistical set and finally not passing the filter.

Without incorporating any knowledge of the system, the system is taught to distinguish the key types of events. This application can be successfully replaced by a knowledge-based solution.
Adaptive rule based profiling: creating rules in the training period and observing their breaking.
A system capturing the rules of the behaviour leads to inability of the operator to re-define a mission stage or functionality change.

5.2 Misuse detection

Misuse detection, also known as knowledge-based [5] – known symptoms of intrusion are described and being searched for in the network traffic (reliably discovers slower intrusion processes undetectable by Anomaly detection)

Misuse detection methods:
Expert systems: if-then implication rules based on previous knowledge, events might be sequential or of certain class or specific.
The simplest incorporation of human knowledge and experience into the fault detection system. Enables definition of known and predictable internal faults of the system, but is limited in the means of outer effects interference.
Model based reasoning: implies the vulnerability of the misuse from higher abstraction of attack behavioural pattern (what the hacker does).
Comparable to defining hierarchy of functionalities, however, in an overall inverse meaning. The system behaviour can be captured into set of actions that is always performed if healthy.
State transition analysis: detects attack by states of the system the attacker has to reach in order to get to the final state (what are the partial results of hacker's activity).
States of failures, especially those causing chain error reports, can be hierarchically captured using a state transition analysis.
Neural networks: a learning net of neurons, which provides flexible rule keeping.
A neural network, comparable to adaptive rule based profiling, does not enable interference with the knowledge once the network is learned. More about the discussed application of neural networks can be found in Cannady [19].

5.3 Hybrid misuse/anomaly detection

Hybrid misuse/anomaly detection is a logical connection of the two previous in order to gain the advantage from both.

5.4 Continuous system health monitoring

Continuous system health monitoring – observation of abstracted key factors of the system.
Continuous system health monitoring detects changes in the system abstraction and compares to the model. In the means of failure detection, this method is comparable to finding a corrective vector to be applied to the system in order to make it behave like the model. Like anomaly detection methods, this approach does not enable human experience to be used to strengthen the detection ability.

5.5 Comparison

According to Jamil Farshchi [8], rule – based intrusion detection approach, which is comparable to a knowledge base approach [5] to failure detection of the satellite, has a main fault in the inflexibility. New types of intrusions evolve every day, which makes a known set of previously encountered failures obsolete. Keeping such a set up to date is an unreasonable requirement, especially facing the fact a statistical method like Kalman filter can simply detect all the non-typical behaviour of the network and show warnings for each event that exceeds the limit of normality.

However, a satellite system in space can be only influenced by a limited number of well-known situations. Some very improbable situations could be omitted from the given knowledge system due to knowledge base responsible scientist's attention loss, nevertheless, as such highly improbable situation occurs and its absence in the knowledge base is discovered, a new subset of information can be simply added to the working knowledge system for correction. The fact the knowledge base is a ground station tool provides this freedom.

Herve Debar summarises the comparison of knowledge-based [5] and behaviour-based [6] approaches to intrusion detection as following:
Behaviour-based detection suffers the necessity of unneglectable false alarm rate, but provides the capability to discover unknown intrusions. However, in its flexibility it might be inflexible from the point of view of changing the activity pattern for known reasons. Also slowly progressing attack can be even considered normal and incorporated into the behavioural scheme.
Knowledge-based detection leads to minimum of false alarms and provides the potential for predictive observations. However, an attack that has not been experienced previously can remain undetected. Also creation and maintenance of the knowledge base is a very demanding task, as new information needs to be incorporated daily.
Using the parallel to the satellite failure detection, unpredictable inner failures are easy to be excluded. However, certain failure combinations disable the detectability of single events, leading to inclination to a knowledge-based solution, which provides a tool for such case. The satellite failure detection topic is significant for:

- Sufficient knowledge of the system
- Sufficient knowledge of the expected inner behaviour and its changes
- Sufficient knowledge of possibly occurring inner failures

These points necessarily lead to the choice of the knowledge-based solution over the statistical. However, the issues of knowledge-based solution remain in the uncertain field of outer failures possible. Unlike the statistical approach, incorporating the outer failures into standardised reactions, knowledge approach offers the feature of defining not only contexts of known failures, but also contexts of known functionalities, thus leaving almost no space for an undetectable failure. However, the complex problem of satellite failure prevention, detection, identification and recovery is more demanding for a single method to cover the entire set of possibilities and needed functions. Hence an additional method has to be provided to cover the outer failures that are impossible to detect by means of knowledge-base.

6 Method evaluation

6.1 Applicable methods

In this part the applicable methods are summarised and in the next one an evaluation is provided.

System diagnostic model of the satellite:

1) basically a matrix of functional (1) and non-functional (0) devices onboard, simple path finding algorithm for breaching the failed part. Advantages: immediate identification, immediate recovery Disadvantages: impossible prediction.

2) If not a 1-0 model, a model expecting and providing more complex behaviour is requested: a matrix representing the state as it should look and a matrix representing the state as it looks. Error detection: difference. Recovery: finding a vector to multiply the faulty matrix with. Issues: very complex model, model faults robustness, modelling each device (is it possible?) Leads to statistical methods.

Statistical methods: Kalman filter related method, making a choice between failed system and working system. Once the filter starts to converge to the failure state, the failure warning is triggered. Advantages: predictive once convergent, Disadvantages: unlike previous case, model abstraction of the sat is needed, long convergence time, no possibility for experimental setting of the probability matrices, delayed reaction at step changes in improbable cases. Identification?

Neural network solution: Assisted learning neural network. Might use the knowledge base, but how? The question stands, is this case of data uncertain or too complex for a classical computer approach? The cost for making the complex task easier for human is the high risk of wrong learning, leading to a risk of mission loss. Detailed analysis of the similarities in satellite subsystem hierarchy is necessary to answer this question. Neural networks learns online -> SW would learn as the satellite is operated = too late. How to explain mission changes to the network? However, we can learn from the neural network programming methodology (weights on the connections would correspond to probabilities of certain information in our database is real).

Other learning algorithms, based on classical programming approach, not neural nets, can be considered highly useful. A decisive tree for diagnostic based on knowledge base, for example.

Knowledge based method: A context definition for each failure and functionality, including Environment and system properties, state and changes, hierarchically preceding and consequent failures and functionalities, all of that necessarily or possibly occurring. Prediction and detection. Hard work making the knowledge base, but we can consider the subsystem patterns and try to generate a systematic solution.

6.2 Issues and cons of approaches

The applicable approaches summarised in the previous paragraphs have several negative aspects or unanswered questions that need to be solved before the implementation. Some cons of the methods, as will be revealed, are not suitable for the given problem at all.

6.2.1 State diagnostic model

How to model the entire satellite and the units?

Can the state matrix be a part of knowledge base?

6.2.2 Statistical methods

How to compensate for outer influences unmodellability?

Why to create a redundant approach to the one used on-board?

Statistical methods are a well-known, widely used, powerful tool in 4 of 5 requirements the task demands. However, the most important part, facing rather unmodellable interferences with human errors, environment changes, outer AOCS related failures or multiple errors, is not covered, as statistical methods only process the information about the satellite system and physical behaviour.

6.2.3 Knowledge-base

How to connect the solution with one providing simple case recovery?

How large will the resulting database be?

How to provide complete process information regarding outer influences?

How to implement the knowledge base architecture?

Knowledge based solutions nowadays encounter several relevant management demands, mentioned by Mattos [16]:

- Efficiency of management for large scale bases
- High reliability and robustness towards memory media loss
- Distributed knowledge base problem
- Knowledge independence

6.2.4 Neural network

How to make it learn on ground before the mission?

How to assure the learning was right for every possible case?

How to secure mission phase changes are known for the net?

How to provide 10^{3000} years of standard telemetry data for learning?

Stergious and Siganos [3] define the purpose of neural nets as following (cited in *italic*):

"Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques."

Deducing from such definition, it is a technique barely suitable to satellite FDIR case, as precise data and capability of orientation in them is present. Extracting patterns ability offers a platform for creating the knowledge base for a traditional computation algorithm, however, only in case it can not be done more reliably by a human.

"The problem solving capability of conventional computers is restricted to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do."

However, the problem stands in human expert having all the needed knowledge.

"Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable."

Creating a knowledge based system of examples is unreasonably complicated task leading to barely the same solution as the knowledge base provides by itself. If the examples are present due to mission duration, the lower reliability of the solution leads to neural net technique discrimination.

"*On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. … These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.*"

As the expert knowledge provides the algorithmic solution and the requirement of predictable behaviour is obviously stated, conventional solution is necessarily preferred by the neural net experts.

"*Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks.*"

Exactly following this statement, satellite FDIR task is suitable for the classical approach and on the contrary.

A thought might occur, that medical application of neural nets, so comparable to satellite FDIR application, is progressing. However, a major difference discriminates between the two utilisations. The expert knowledge provides the algorithm, not the examples, not even taking properly chosen examples requirement into account.

6.2.5 Inductive Logic Programming

How to interconnect outer influences knowledge with system knowledge base?

Inductive Logic Programming offers an alternative to the non-applicable neural net, however, still remaining within the necessary field of artificial intelligence. Formulating algorithmic descriptions of usually encountered problems with human errors, environment changes, outer AOCS related failures or multiple errors can, however, be a rather demanding, long term task.

6.3 Requirements and performances of approaches

System diagnostic method: huge matrix of all the functional parts onboard, redundant parts and their states (0 – unavailable, 1 – working, 2 – ready power on, 3 – ready power off)

Requirements reachable: 100%

Performance provided: Detection, Isolation, Recovery, 50%

Final decision: APPLICABLE AS A TOOL

Replaceable by other solutions: 100%

Statistical methods: State space representation of the whole system, comparable to the satellite output, experimentally determined probability matrices, interfering influences behaviour model.

Requirements reachable: ~75%

Performance provided: Prediction, Detection, Isolation, Recovery, 66%

Final decision: APPLICABLE AS A TOOL

Replaceable by other solutions: 100%

Knowledge base: context of each failure and functionality OB + part hierarchy + outer influences knowledge + operator monitoring

Requirements reachable: 100%

Performance provided: Prediction, Detection, Isolation, Recovery, partially Outer influences interference reactability, Operator check, 91%

Final decision: APPLICABLE AS A TOOL

Replaceable by other solutions: 50%

Neural net: large amounts of data from a simulator or previously running mission, comparable to 10^{3000} years of mission to reach 80% probability the system has learned.

Requirements reachable: <1%

Performance provided: Prediction, Detection, Isolation, Recovery, Outer influences interference reactability, 83%

Final decision: NOT APPLICABLE!

Replaceable by other solutions: 100%

Inductive Logic Programming: knowledge base covering the system of the S/C, mission and operator + algorithmic descriptions of impacts the outer influences can have

Requirements reachable: 100%

Performance provided: Prediction, Detection, Isolation, Recovery, Outer influences interference reactability, Operator check 100%

Final decision: APPLICABLE!

Replaceable by other solutions: 0%

6.4 Suggested solutions

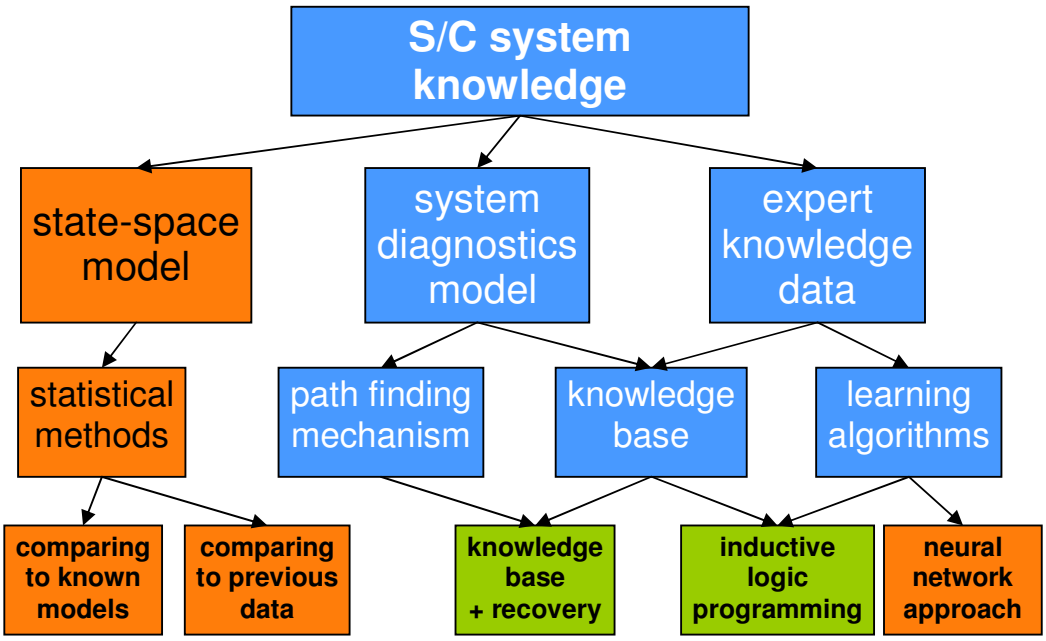


Fig 6.1 Chosen solutions

Neural network using a knowledge base in a fashion. Proven **unreachable** according to Valiant [15]. Leads to ILP.

System diagnostic model providing detection and basic recovery, interconnected with knowledge base, providing prediction, identification and partial outer interference handling.

Statistical data processing connected with a knowledge base. Leads to redundant data processing and incomplete coverage of the problem. **Unsuitable solution**.

Inductive Logic Programming (connected with a predictive method, Kalman like predictor for example). Knowledge base connected with learning algorithm.

7 Venus Express failure categorisation

Venus Express is a project of Venus observation. On November 9th of 2005 it was launched from the Baikonur Cosmodrome in Kazakhstan. After several orbits around Earth the satellite gains the sufficient energy for a trip to Venus from an Earth bypass. Successful insertion into Venus' orbit took place on April 11th 2006. Venus Express achieved its desired orbit in May 2006 and has been sending routine data from its science instruments. The mission is expected to provide data on the Venus atmosphere for several of the planet's days each. The expected end of mission is May 2009.

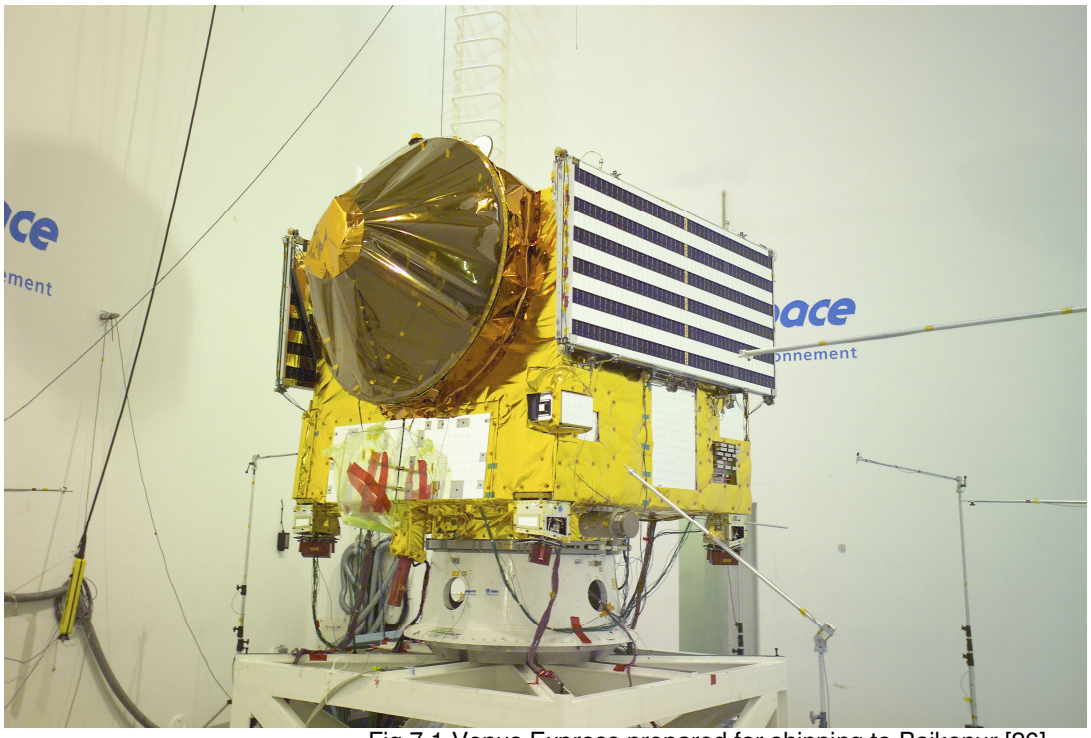


Fig 7.1 Venus Express prepared for shipping to Baikonur [26]

7.1 FDIR System onboard

The failure handling system onboard of Venus Express (Fig.7.1) has two distinguished layers, a SW and a HW one. The most sensitive subsystem is AOCS (Attitude and Orbit Control System), followed by Power and Thermal subsystems, as these are vital subsystems of the satellite. The onboard FDIR has several specific modes to contain the faulty situations. The main FDIR strategy is as follows:

1) Mission integrity

2) Mission continuation

3) Ground intervention between 2 error events expected

4) Minimum reconfiguration necessary

The points 1 and 2 are the same in the phase of Venus orbit insertion. Every unit is defined as nominal or redundant in order to distinguish the original setting. On the ground station software, a Redundancy table is maintained in order to keep track of these two statuses of the units. The FDIR has five levels of autonomous reactions.

The most significant of them is the Safe mode, a mode caused by severe subsystem damage, leading to a reconfiguration and reboot/restart of the systems, followed by Sun Acquisition Mode (SAM) in order to recharge batteries while waiting for ground commands. A SW and HW Safe modes are recognised. Software safe mode is a reboot of the system, whereas hardware inhibited safe mode means stopping the DMS processor, which triggers the watchdog reconfiguration modules.

If the damage is not as severe, a simple reconfiguration is triggered, bypassing the set of suspected units by a set of redundant units. The summarisation is:

Level 1: after the reconfiguration the mode is unchanged

Level 2: suspected units reconfiguration followed by SAM (SW Safe mode)

Level 3: SAM followed by another surveillance leading to a global reconfiguration and another SAM with all the units switched to redundant

Level 4: after the preceding level, another surveillance is pending, leading to HW safe mode being triggered by DMS (PM reboot), followed by SAM with all nominal units used

Level 5: after the level 4 procedures fail to contain of the situation, another PM reboot is triggered with all redundant units being used. The further surveillance is inhibited, along with the SAM – SHM (Safe/Hold Mode) transfer.

These 5 levels are supposed to cover for every situation possible, other recovery procedures are to be handled by the operator.

Also, in order to prevent a SEU (single event upset) caused reconfiguration, a filter is applied onboard to the relevant data. Therefore no surveillance is triggered based on a random, one-time malfunction of a sensor.

7.2 Telemetry

The Venus Express telemetry includes 28830 parameter blocks that can reach several levels in value. It is a large set of data, however, every bit has its meaning.

Using the appropriate queries, housekeeping data can be separated from other, failure detection irrelevant information. In the database of telemetry data, a column named SERVICE provides the information on which system is the data related to. For FDIR, the service values of 3 and 5 are used. Using this filter for incoming data, along with the unique values requirement enforcement, the number of parameters (with all value levels they can obtain) is eliminated to cca 6480. Further elimination with respect to certain subsystem is possible as well, leading to an ease in the implementation of a proof-of-principle program, providing a one-subsystem solution.

The telemetry data varies in the frequency of updates, for example Thermal relevant data is sent every 256 seconds, whereas Status comes in the intervals of 4 seconds. An agreement exists, that every downlink frequency will be given by a period of 2^n seconds in order to prevent confusion, however the operator has the ability to change the given frequency if necessary. It is possible to perform a database update every time a new relevant packet is detected, as the Mini Control System already delivers not only TM Packet History, but also packet counter, which provides the information on whether a packet related to given table cell has been updated or not.

As mentioned in 7.3, the Mission Control Systems already provide telemetry pre-processing mechanisms and deliver data that are more comprehensible and easy to process further. For example, for each device, its each mode and each mission stage, there are different limit values for situation criticality. In the pre-processed database, these boundaries are provided along with the real values. Most of the parameters have 4 boundaries, naturally lower and upper limit, but also distinguished into soft limit (warning is raised, but the situation is not

critical) and hard limit (at this value, there is no guarantee the device continues to work). Some parameters need no soft limits or even do not need any limits, as their value is more like a binary flag. These can be distinguished due to the TMPA-SIZE parameter, which tells the data size of the boundary values and DTYPE, which provides information on the number of limits by its value. For the limit exceeding values of parameters, the LIMIT flag is raised to 1, which makes the critical data observation triggering an easy task.

7.3 Tools available

As mentioned earlier, there are several on-ground tools that might be of help. As the new tool is not intended to re-invent things or to run redundant functionalities, these tools are to be examined before the start of the actual work. Firstly, the Mission planning tool, providing the mission planning from a rough outline during the preparation down to fine-tuning of the attitude in every second of the mission. The MP provides warnings for intended manoeuvres that might endanger any of the sensitive devices (e.g. pointing a star tracker into the Sun). However, such protection is but one part of the knowledge that has to be involved in the overall warning system.

Another useful tool is the Sun vector tool, providing information on the direction to the Sun at every moment. The use of such tool is obviously in improving a fine-limit setting for temperature sensors onboard. However, implementing the use of this tool into the FDIR procedures would require detailed knowledge on device placement onboard and therefore will not be considered in this work.

Before obtaining the data from the satellite simulator, certain pre-processing is done by another tool, Mini Control System built on Satellite Training Centre, providing translation from a stream of hexadecimal digits into real values, calculating the current limits and providing even the information on which packet updates which cell of the table and how often.

A useful tool is also a Pre-transmission Validation (PTV), which stand between the operator and the satellite and provides warnings on intended actions, which are impossible or harmful. However, this tool doesn't provide a sufficient knowledge and deserves to be extended.

7.4 VEX propulsion subsystem

The propulsion subsystem is the one that has been chosen for a proof-of-concept software tool, created as a part of this work.

For Venus Express the propulsion subsystem is based on a conventional bi-propellant technology, and the propulsion subsystem schematic is unchanged from Mars Express. For the Venus mission the total propellant mass is increased up to about 530 kg compared to the 480 kg mass on Mars Express, this increased mass is still inside the tank qualification heritage.
The propulsion units are the same on Venus Express and Mars Express with just one exception. The pyrotechnic valves are of a different type. However, a problem appeared during the Venus Express mission, as a specific kind of explosive was planned to be used and tested and another kind was applied in the final stage. The untested explosive is of higher strength, therefore pyro-valves are an issue on the VEX mission.

However, the use of the propulsion system is fluent. It consists of the following units (fig 7.4):

- The High Pressure Pressurant Control Assembly, composed of :
    - One pressurant tank (HE)
    - One high pressure fill and drain valve (FDV1)

- One high pressure transducer (PT1)
- One normally open pyrotechnic valve (PVNO15)
- Two normally closed pyrotechnic valves (PVNC1 and PVNC2)
- One pressure regulator (PR)

- The Low Pressure Pressurant Control Assembly, composed of:
  - Four Non Return Valves (NRV)
  - Two low pressure fill and vent valves (FVV9 and FVV8)
  - One low pressure transducer (PT2)
  - Two Low Flow Latch Valves (LV1 and LV2)
  - Four normally closed pyrotechnic valves (PVNC3, PVNC4, PVNC5, PVNC6)

- Two Propellant Control Assembly, composed of:
  - One titanium propellant tank × 2 (NTO and MMH)
  - Two normally closed pyrotechnic valves × 2 (PVNC7 and PVNC9 for NTO1, PVNC8 and PVNC10 for MMH2)
  - One propellant filter × 2 (F1 for NTO1 and F2 for MMH2)
  - Two low pressure transducers (PT3 for NTO1, PT4 for MMH2)

- The Prime and Redundant Thruster Assembly:
  - 2 × 4 Flow control valves (FCV)
  - 2 × 4 Thrusters Latch valves (TLV)

- The Main Engine Assembly:
  - Four normally closed pyrotechnic valves (PVNC11 and PVNC13 for NTO1, PVNC12 and PVNC14 for MMH2)
  - Two normally open pyrotechnic valves (PVNO16 for NTO1 and PVNO17 for MMH2)
  - One Nominal and one Redundant Flow Control Valves (FCV)

Venus Express has a bi-propellant propulsion system. That means, two propellant tanks are onboard, one for oxidiser (NTO) and one for fuel (MMH). VEX has one high specific impulse Main Engine (414 N, fig 7.2), used for all the large trajectory corrections and 4 Nominal and 4 Redundant 10 N thrusters (fig. 7.3) to complete the system for attitude control, finer trajectory corrections and Reaction Wheels off-loading.

Fig 7.2 The main engine

Fig 7.3 10-Newtons Thruster

High pressure helium is stored in a single 35.5 litre pressurant tank. The pyrotechnic valves are recognised as normally open (N/O) and normally closed (N/C) valves. The normally open can be closed and normally closed can be opened by a small explosion, causing the valve either to block or to unblock, thus why they are recognised as pyrotechnic valves. The normally closed valves are opened during the propulsion system initialisation. The unidirectional valve only allows the flow in the direction of the arrow and flow control valve is what allows the thrust intensity to be influenced. Low flow latch valves are valve equivalents of switches and at least one of the redundant valves is open in normal regime. The pressurised helium from the helium tank is used to push the fuel and the oxidiser from their tanks, using a membrane. After the launch, the initialisation starts by opening the normally closed valves under the helium tank, which causes a rapid change of pressure inside the system, followed by equilibrium state. Subsequently, other valves are opened in order to pressurise the bipropellant tanks and finalise the initialisation. The blue flow control valves are certainly not pyrotechnic, as it is crucial to control the opening of these. The regulator controls the downstream pressure in the propellant tanks and liquid lines to 17.5 bars.

The pyro valves are placed for protection of the fuels and helium reserves, as in case of flow control failure, the precious contents of the tanks has to be saved from leakage. Every pyro valve has two pistons to close/open it, controlled by explosives of two different dates of production in order to assure no accident can happen when the firing is required. For safety reasons, all thrusters are in hot redundancy.

The power to the pressure sensors is delivered through switch LCL14B or redundant switch LCL14A. The current through these switches is measured in order to detect a failure. The temperature is measured on all four pressure sensors, all 18 flow control valves and on several places in each of tanks.

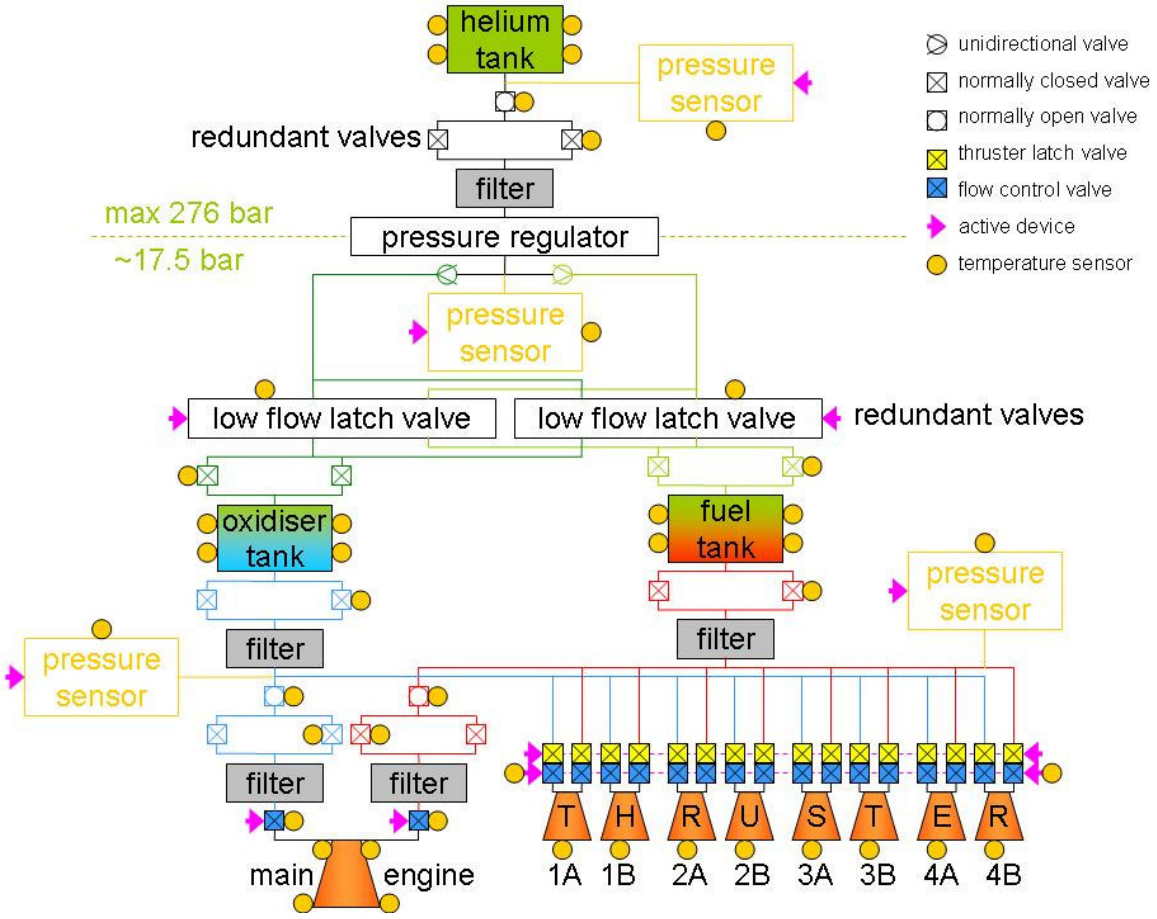


Fig 7.4 Propulsion system of Venus Express

7.5 VEX propulsion subsystem relevant TC/TM

To detect the relevant telemetry and telecommand for the given subsystem in the normal traffic is not an easy task. First, the simulator telemetry database, consisting of 28829 parameters was processed to obtain the proper packet header names, using also the assistance of skilled experts Joachim Ochs and Carol Quirke. A good idea is to realise, that the given subsystem only has telemetry values measured in bars, degrees Celsia and mA (the power supply to the subsystem) and simple Boolean on/off or true/false expressions. This leads to a useful query, filtering only these parameters that are expressed by such types. Also certain patterns are useful to observe in the parameter name, like for example strings SAS, SADE or RW are relevant for Sun acquisition system, Solar arrays and Reaction wheels, respectively. When considering all the possibly measurable values within the system, a final number of parameters to observe can be obtained. In total there are [Appendix A] 4 pressure readings, 58 temperature readings and 38 active devices, which are powered through a switch that is ON or OFF and has a value of current coming through.

8 Algorithm description

Once knowledge is summarised and the solution chosen, what remains is the algorithm fine definition based on the gained knowledge. First, a detailed application theory on Expert system development and Logic programming extension is necessary.

8.1 Theory

As already mentioned in chapters 4 and 6, knowledge base is the necessary component of the solution. Therefore various knowledge base algorithms have to be evaluated and the optimal solution chosen.

8.1.1 Introduction to expert system application

According to Mattos [16], a knowledge system is implemented based upon three major steps: knowledge acquisition, KB structuring and KB evaluation and feedback. Just like human expert, a knowledge base starts by implementing a single piece of knowledge and exploiting this into a bigger database, later improving and broadening the knowledge.

The essential two focal points to be grasped by the knowledge system (KS) development are knowledge and the process of solving problems. Only and only if the KS tracks these two concepts of expert insight of the situation, it can become a successful implementation.

Mattos [16] analyses human knowledge and problem solving capabilities and distinguishes the problem solving knowledge into domain knowledge and problem solving methodologies, which are further divided into problem solving strategies (known algorithms) and heuristics (unknown exact algorithm approximation or known complicated algorithm simplification).

The resulting system should:

- support the acquisition of the new knowledge
- assume its storage and management
- apply it to solve problems
- explain the solving process

Typically, knowledge base, grasping the knowledge domain part of the algorithm, includes the domain knowledge, the current state of the problem and its solution and finally heuristic rules of information processing.

The gradual increase of the KB volume is explainable by the fact that every inference result is stored in the knowledge base and every additional piece of information is hence put into relations with the information available previously, thus the growth of the knowledge base is approaching rather exponential behaviour than linear.

The problem solving component (inference engine) should consist by a cognitive program, creating and checking hypotheses, and an additional control mechanism, ensuring the inference engine does not generate unwanted or unimportant conclusions or steps leading to them.

The system interfaces are designed for 3 groups of dialogs. Firstly, the knowledge base engineer providing well-formulated knowledge on one side and the data acquisition tool on the other side. Secondly, the explanation component providing information about the reasoning from the first impulse until the conclusion. And thirdly, a user-dialog, providing the final conclusions in a user friendly form to a user without deeper knowledge of the problem.

8.1.2 Knowledge base development

The development of a knowledge base starts with a rapid construction of a small prototype, usually in presence of a knowledge engineer and an expert, followed by performance observation and comparison to a human expert behaviour. After the performance is comparable, the prototype is put into process of knowledge broadening.

Being already run as ready expert software, the knowledge system is provided a constant feedback in order to **acquire** new knowledge, **incorporate** it into the knowledge base, **applying** it subsequently and **explaining** the new steps.

The previously loosely described procedure can be identified as the following 4 step sequence (fig. 8.1):

Knowledge acquisition – gathering domain knowledge and generating heuristic rules

Expert system design – creating a knowledge representation scheme, the inference engine and implementing problem solving strategy

Knowledge modelling – knowledge base implementation

Knowledge refinement – heuristic rules and knowledge revision based on feedback

The key product of the knowledge engineer interference with the human expert is the problem solving heuristics, as any other information within the field can be obtained from literature. Human experience in problem solving is irreplaceable.

Subsequently, the optimal solving strategy is chosen, as will be discussed later, and implemented. This prepares the situation for the knowledge insertion. After the introductory steps, the system is improved by knowledge refinement, leading an imperfect prototype into expertise. Figure 8.1 provides the schematic understanding of above mentioned [16].

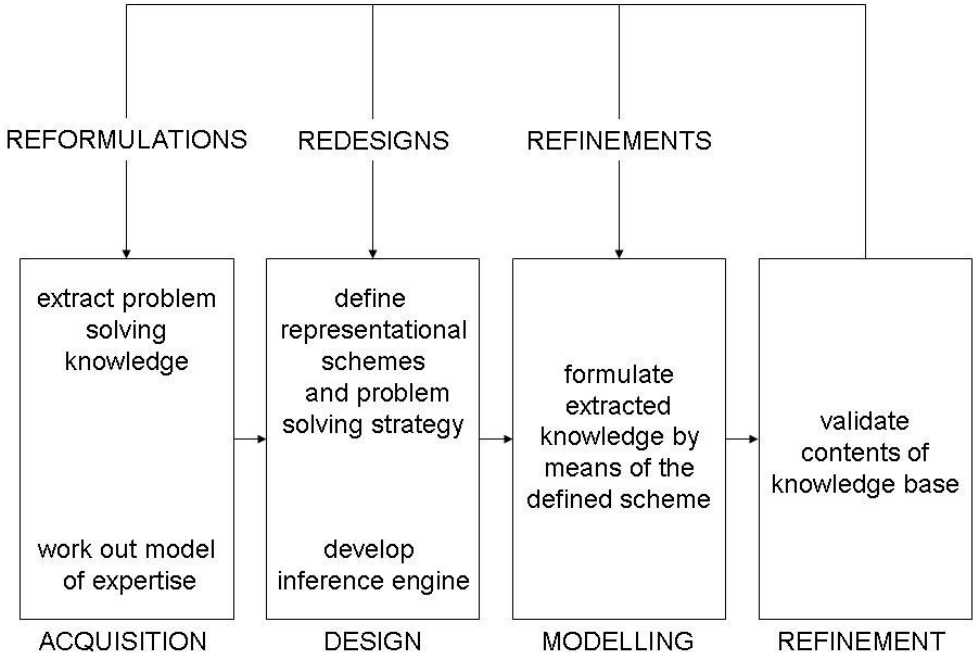


Fig. 8.1 Evolutionary process of expert system development

The contents of knowledge base can be distinguished into three types: **declarative** (passive) data, **behavioural** (active) data and **structural** (organisational) data. Therefore, understanding of the world through the view of an expert system can be divided into four phenomena:

Entities – objects that exist. Usually have some properties (frames of properties), can be classified into classes, subclasses and superclasses and have some dependencies on other objects.

Roles – generic entities becoming specific in the meaning of some action or event. Object dependencies are described by roles.

Actions – events that happen to objects, being caused by objects or between objects. Every action has preconditions and consequences.

Situations – states of the world in certain time instants. Values of all the properties of all the objects involved in the observed system.

Independent of these types, certain definitions are valid for any kind and form of the knowledge base. A summary of true statements from the knowledge base can be interpreted by so called **models** of the base. An **inconsistent** knowledge base is defined as a base inducing no model, e.g. having contradictive statements. The fact an inference scheme only deduces true sentences from the base is called **inferential validity** or **soundness**. The knowledge base is said to be **complete** if every statement expressible by the inference engine can be derived from the knowledge base contents.

From the declarative knowledge base structures is the semantic network of our interest and procedural schemes provide a very useful model of production systems.

A Semantic network structure interconnects all pieces of information by certain relations, as shown on fig. 8.2. A Production system is a set of IF-THEN rules, describing meanings and actions. For such semantic structure as the one on fig. 8.2 a production system could be:

if solarcell_1_b is_faulty then

(report(decrease(functionality(solararray_1_b)))

and if functionality(sun_sensor_1) < limit

then recommend(switch_on(sun_sensor_2)))

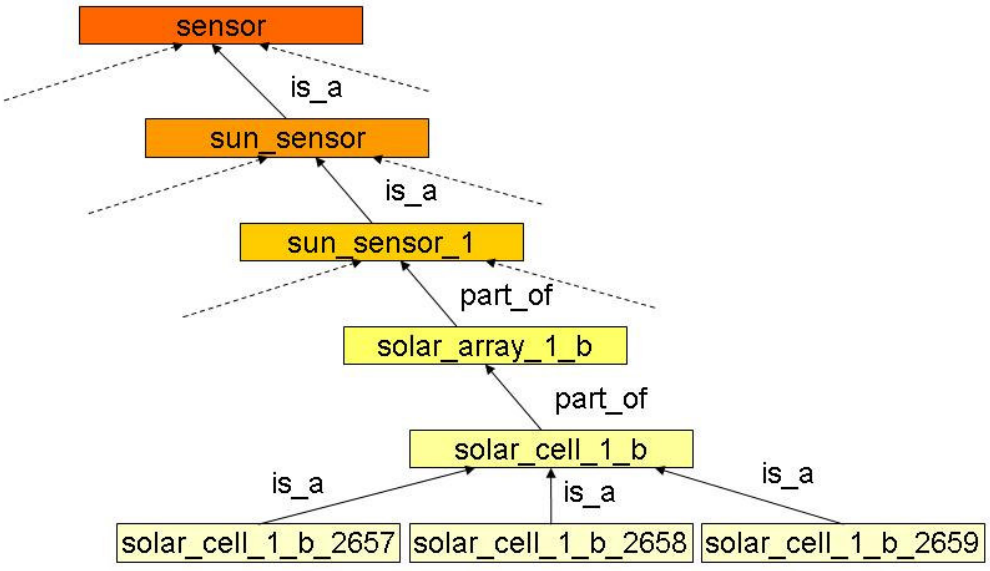


Fig 8.2 Semantic architecture

However, as given problem is very complex, a frame structure is to be considered as well. A frame is a set of information related to a stereotypical situation (a failure has a consequence), organised into a block structure (failure contexts). Such frame can then be applied to multiple situations, providing coherent contents of the knowledge base.

Taking advantage of the Data-oriented paradigm and the Production system, we achieve a coherent knowledge base, which is dynamically updated and maintained in order to reflect the circumstances of the real world.

8.1.3 Inference engine

The **inference** is a word used for the reasoning, drawing conclusions or generating new knowledge. The problem is obviously in the difference between human and computer approach to the knowledge processing. Unlike human, a computer has no preferred rules, as no rules “seem” more likely to lead to the conclusion.

There are several strategies, how to approach the reasoning. Either **data-** or **goal-driven strategies**, choosing cyclically a random rule, whose condition is fulfilled, to reach conclusion and compare it with the wanted one, or on the contrary, or so called **conflict resolution strategies**, attempting to determine the right rule to be applied.

The data-driven strategy, also called forward, bottom-up or antecendent reasoning, is a method reaching from the conditions to the goal by trying all applicable rules, independently of the relevancy of those. If is of use for diagnostics, as it implies all possible failures from the symptoms.

The goal-driven approach, also called backward, top-down or consequent reasoning, is the opposite. Using this method, a desired goal is chosen and rules applied backwards in order to achieve the conditions and compare them to the actual state of the system. This approach creates less irrelevant branches of possibility.

The search strategy is a method applied to either forward or backward reasoning. It either supports building the entire branch possible first (depth-first) or builds all possible new elements in all the branches at the same time (breadth-first). However, also heuristic search is available, putting weights on the rules according to their probable usefulness.

Conflict solving strategies are used to determine the very rule to be applied first or a sequence or set of rules to be preferred or discriminated.

Unlike inference, a **problem solving strategy** is a method taking the problem stated into account and therefore is task specific.

8.1.4 Structure

As mentioned in 6.3, nowadays the possibilities of knowledge based solutions are broadened by ILP in such manner that makes the bare knowledge based approach cover only certain part of the whole set of actions provided by ILP approach. Nevertheless, even in the knowledge base domain, Spacecraft and Mission knowledge is but one part of the knowledge existing. A very crucial task is to handle operator failures and interactions of the spacecraft and the outer environment, that might be extraordinary and in a fashion, unpredictable. Merely monitoring the inner S/C system and its relation to the mission is 2x more than an analytical approach allows, however, there is still space for improvement.

Therefore there are four knowledge domains to be implemented, necessary for a sufficient tool:

Spacecraft Systems Model – a hierarchical description of the inner system of the S/C and its components; using the engineering knowledge of the satellite system, its subsystems and their related behaviour, the FDIR can be performed

Mission Model – a hierarchical description of the mission, its phases and their subparts; using the knowledge of the mission model, phases, procedures and TC/TM, the resource evaluation can be performed. The main issue of this part is a constant implementation of the new knowledge from the fine mission planning.

Functional Model – connecting engineering and mission related data in order to provide trend analysis and failure prediction, detection, identification and recovery recommendations to the operator, as well as to discover and describe possible failure propagation

Operator Behaviour Check – based on functionality hierarchy knowledge and device state monitoring; in order to prevent human error related failures, when an action is required by the operator, a check is triggered to generate warning in case such requirement is impossible or endangering the S/C. The possibility check simply compares the preconditions of a command with the current state of the satellite. The endangerment check is nothing else than a simulator, running an intended operation and providing feedback before the command is actually applied.

As an extension to the knowledge base a very important part derived from ILP theory is tailored on the knowledge base, a **set of algorithmic description of known possible extracurricular failures and states**, both spacecraft and mission related. The issue of this module is the ability to accept new algorithmic descriptions of the yet unknown failures. The fashion of definition of knowledge base allows us to consider and name this part a knowledge base as well; however, the major principle difference in the knowledge incorporation is to be pointed out in this work.

8.1.5 Logic programming extension

Inductive logic programming is a synthesis between a knowledge based inducing expert system and an algorithmic approach, traditionally leaving the knowledge implicitly in the code. Obviously, this synthesis is not difficult, as the use of knowledge base can be performed by several processes, running in parallel. For example, at the same time an inference engine of the mission production system changes the boundaries for failure suspicion with regard to a new mission stage, a comparator makes note of a valve temperature change, a simulator performs operator check and a specific failure check is triggered by a set of failures, which might lead to its conclusion.

8.2 Practical algorithm description

8.2.1 Failures

In order to understand the meaning of algorithm definition in chapter 9, an overview of failure types, as well as the simple algorithm of their identification is necessary. Several understandings of failures need to be taken into consideration.

Firstly, a common sense classification:

- phantom (pseudo failures), caused by either
    - mis-configuration of the OBSW (can be solved by a new OBSW upload)
    - operator mistake like
        - wrong action (can be solved by operator correction)

    - situation misunderstanding (can be solved by operator support)

- time caused degradation of materials (usually happens after 10 years of mission)
- specific HW failures, simple and easy to pin out
- unmonitored failures or their combinations, allowing merely a monitoring of their outputs

Another classification is derived from the overall understanding of errors, not the experience related approach:

- process – something is being done wrong
- functional – some functionality is not fulfilled
- HW – some specific piece of hardware is damaged

8.2.2 Failure recognition methods

In this point, an algorithm becomes explained. For every failure possible within the system, certain context (mode [22], frame [16]) can be defined, meaning a set of necessary conditions and consequences the given failure has. The same can be done for functionalities, leading to a simple algorithm of decisions:

- If the necessary settings conditions for the functionality are fulfilled, it is not the process failure.
- If the functionality consequences are not present, it is a functionality failure.
- If the error consequences are fulfilled, it is a HW failure.

In other, more algorithmic approach:

0) Check conditions necessary for failure occurrence.
    a. YES -> it is an error
    b. NO -> it is not an error (it is a phantom error)
1) Check settings necessary for functionality state.
    a. YES -> it is not a process error, it is a functionality error
    b. NO -> it is a process error
2) Check HW preceding and subsequent failures in hierarchy for identification.
    a. Preceding fulfilled -> move up in hierarchy, likely NOT an error here
    b. Subsequent fulfilled -> HW error
    c. Subsequent not fulfilled -> simply weird functionality, repeating the command recommended

The failure identification can be complicated, however, solutions are available from the following points of view:

- Trivial cases the only require detection and failure mode analysis
- Undetected failures are detected through methods experienced by operators or by simple functionality mode analysis

- Multiple errors are possible to discover as chain reactions (hierarchical behaviour) or random coincidence (where the question stands which combinations are relevant to take into account)

Failure prediction requires a simple observation of values that exceed their limits. As this information is provided in the telemetry already, the observation can be triggered by “out of limit” warning. Such observation would consist of applying a low pass filter on the relevant value and calculating a trend, which leads to an ability to predict the intersection between the predicted evolution of observed parameter and its critical value level.

8.2.3 Subsystem specific requirements

Points of interest are as follows:

- pressure and temperature readings should be corresponding to:
  - mission phase
  - telecommands performed
  - each other
  - other readings within the system
  - the hot redundancy readings
- the sensors should be checked to have:
  - power
  - redundancy
  - value reading

These requirements require reading and processing of the following data:

- telemetry
  - power subsystem parts related to the valve and tank sensors
  - position and attitude data
  - sensor and hot redundant data from the whole subsystem
- telecommands

The reading of the data requires careful filtering of telemetry and telecommand. From the Venus Express simulator specifications it becomes obvious that there are only very few packets to be observed and that all the tank temperature sensors, for example, are powered and read data from by one node called RTU, Remote Terminal Unit, therefore once the power is provided to RTU, all the tank temperature sensors can be considered powered.

8.2.4 Low pass filter design

The incoming telemetry data requires a low pass filter, as the sensor data are noisy due to many factors. Reasonable solution is to group the sensor data according to their overall behaviour and set the low pass filter for each group, for example, thruster temperature sensors are expected to encounter rapid reading changes in short time periods, therefore a less sensitive filter (up to 30 last values) is suitable, whereas the pressure sensor temperature readings are expected to stay strictly within the limits, as higher temperature might damage the sensor, therefore a more sensitive filter, working with cca 5 last values' average, is to be used. Slight differentiation between the pressure sensors and the temperature sensors filter is desirable as well, as the relation between a usual value and its boundaries differs.
A proper extrapolation method has to be used as well, which is to be applied after the filtering. Fortunately, the choice of the method is heavily influenced by the fact that very few situations really occur in space, leading to our ability to predict the data behaviour for each of them. The polynomial extrapolation appears to bring the results approaching the real data behaviour for the possible cases. Obviously, some balance has to be found, providing an extrapolation method as well.

The most common methods of extrapolation are Linear extrapolation, Conic extrapolation and finally Polynomial extrapolation. The linear extrapolation provides an extension of the trend line, assuming the behaviour of the system can be considered linear-like. The conic extrapolation uses conic sections as patterns to be observed on the data behaviour, leading to a more probable extrapolation in many cases. The Polynomial extrapolation creates an addition of polynomial curves, fitted to the observed set of points, leading to very precise estimation in cases, when the behaviour really is polynomial-like.
However, the pressure and temperature behaviour can be described very accurately by a set of linear or linear-like trends, without the risk of reliability loss. A nice implementation is the method of triangles, calculating the centre of masses of the last three values, thus obtaining a smoothed curve (low pass filter), easy to be fitted with a linear extension. As this practical approach provides both low pass filtering with an adjustable sensitivity (the triangle size) and a good ground for linear extrapolation, it seems to be the optimal solution for desired filtering module.
To prevent failures and false alarms in the filter, the two-limit property of parameters can be used. Every parameter is delivered with a set of two soft limits (their crossing raises a warning) and two hard limits (the value that is fatal for the system). These limits have been pre-calculated from the knowledge of mission phases and of the housekeeping values considered normal and those not normal, yet still feasible for the normal operational state of the devices.

9 Implementation and testing

As revealed in chapter 8, expert system development stages are:

- Knowledge acquisition
- Expert system design
- Knowledge modelling
- Knowledge refinement

The design part is considering the Expert system design and Knowledge modelling.The implementation of any software project has a well-known lifecycle, starting with user requirements and ending with system testing, as shown on fig. 9.1. The contents of such lifecycle is a drift in understanding the problem from the satellite operator point of view, from the point of view of a knowledge base expert and finally from a programmer's point of view. The steps undertaken on the left branch of the V cycle are broadening and fine-specifying iterations leading from a simple user requirement definition, over the theoretical background of the solution to the actual coding. Certain tradeoffs, corrections, improvements and simplifications are necessarily a part of this cycle.

An important first step is a name of the software product as well, as that is how users and developers refer to it. Of course a name like **MiCoFaPrDIR** (standing for Mission Control Failure Prediction, Detection, Isolation and Recovery) could be applied, however, much shorter and more decent Failure **PreDetIR** or **PreDIdeR** (Prediction, Detection, Isolation and Recovery) are terms more sounding and easy to remember due to pronunciation revoking the word "predator". As a matter of fact, a failure predator is a very fitting name for such project.

The issue of any software implementation is to select the proper tools for such activity. A database system is obviously a necessity for the knowledge base implementation, therefore SQL query system could be chosen as the fastest, most stable possible tool. To handle the inference engine, a well-standardised language with real-time extension possibilities is required. A high level language obviously creates a possibility of re-usability, however, popular Java approach is unnecessarily overlapping the easy task and limits the real-time use options. After considering the environment and the tool already implemented in other languages, the idea of SQL database has been rejected for a high computational cost of the incoming data conversion into 2 different database systems with 2 different engines. Therefore a step towards commercial tools has been made, promoting MS Access database system and its queries and macros, though the performance meets the limitations of this tool in several points, most significantly in the lack of Transaction mode, allowing a proper simulation environment. Fortunately, the VEX simulator is already implemented in MCS. A decision has been made to maintain the detailed design within SQL understanding of the problem in order to provide a possibility of an extension in the future. As in many aspects, even in the choice of tools, this work is supposed to provide an analysis to be re-usable. The final implementation is merely a proof of concept, however, even there space for extensions will be provided on several places.

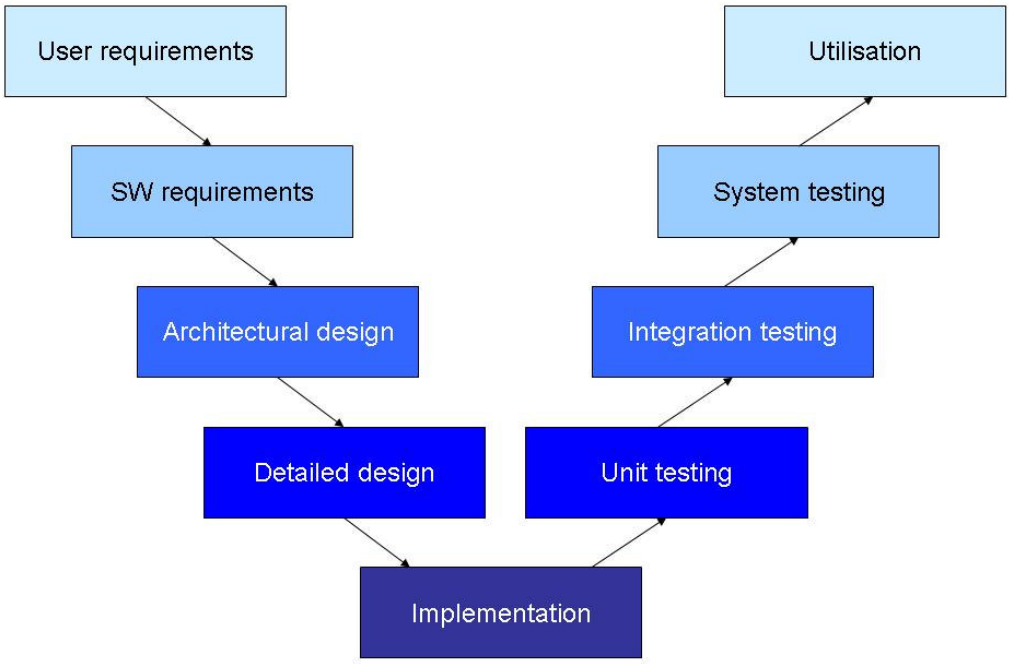


Fig. 9.1 Software V-lifecycle

9.1 Failure PreDetIR development

9.1.1 User requirements

A tool to aid the ground station operator with FDIR, providing also a predictive behaviour and a check of correctness of telecommands is the main objective. A neat interface providing the user with information in case of failure (imminent or approaching) and recommended recovery procedure would be of use, as well as intended action simulator interface.

9.1.2 Software requirements

- knowledge based solution for
- Mission Control Systems,
- providing failure:
  - prevention
  - detection
  - identification
  - recovery hints to the operator
- for failures caused by:
  - inner systems of the S/C (declarative, structural knowledge)
  - outer conditions (behavioural knowledge, logic programming)
  - operator (simulator)
  - unpredictable situations (logic programming)
- tool has to obtain and process data from:
  - Mission Planning tool
  - Mini Control System
  - Pre-transmission Validation tool
  - VEX simulator

9.1.3 Architectural design (fig. 9.2)

- knowledge modules describing:
  - S/C engineering knowledge, declarative and structural
  - mission related knowledge, behavioural
  - S/C subsystem – mission relation knowledge model, behavioural
  - operator action consequences, simulator
  - extracurricular events knowledge, logic programming
- inference engine using following methods:
  - bottom up production rules
  - logic programming
  - simulator run
- user interface
  - graphics
  - identification or prediction
  - recovery suggestion
  - action input

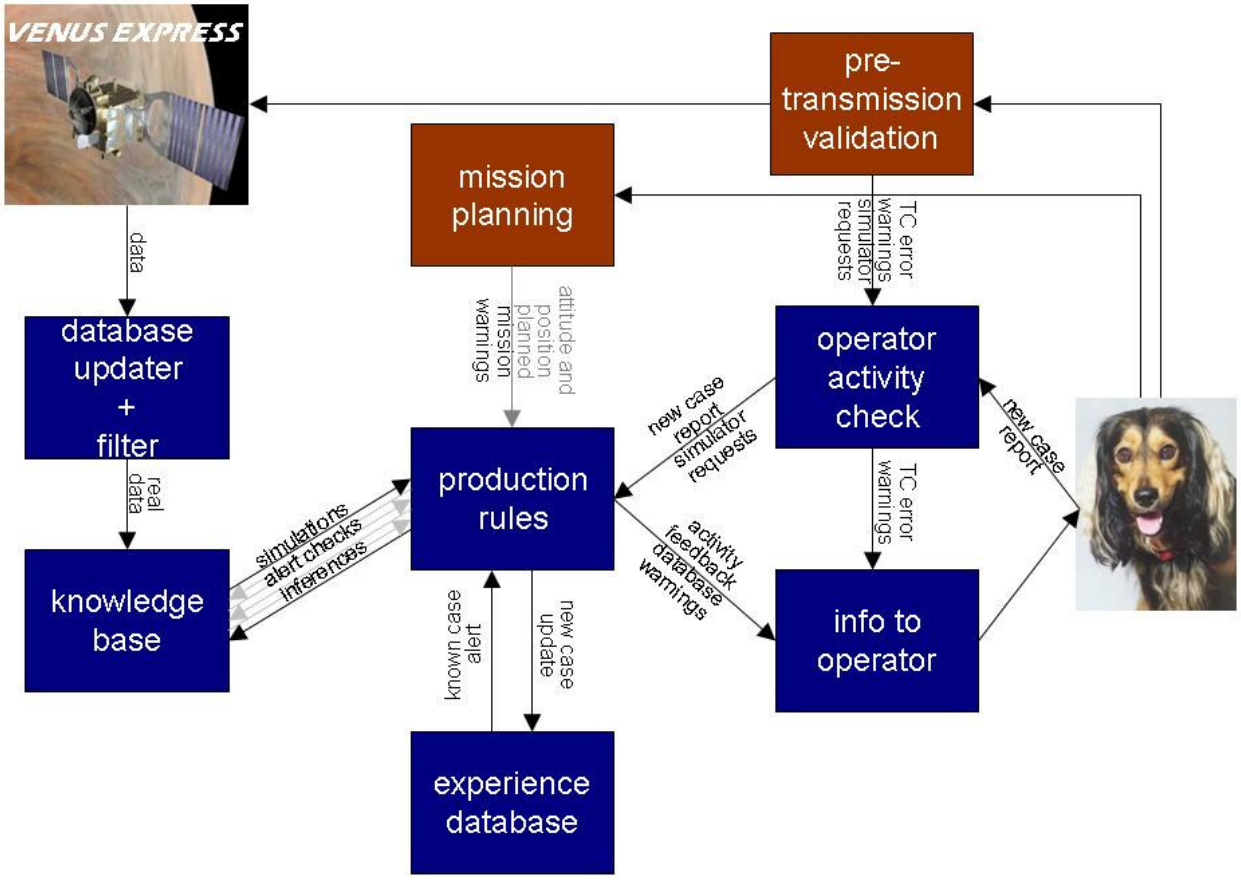


Fig 9.2 Block scheme of the tool

9.1.4 Detailed design

As mentioned in the beginning of chapter 9, each step of the V design is a shift from a wish to its realisation. Architectural design provides the in-view of the knowledge base theory, whereas Detailed design step is merging these requirements into a realistic software module description. Following the schematics on **fig. 9.2 (keywords in bold)**, a detailed description of modules is provided in this chapter, using and explaining the terms provided in the *previous part (keywords in Italic)*.

The **knowledge base** consists of:

- *Spacecraft Systems Model* developed as a set of knowledge holding frames:

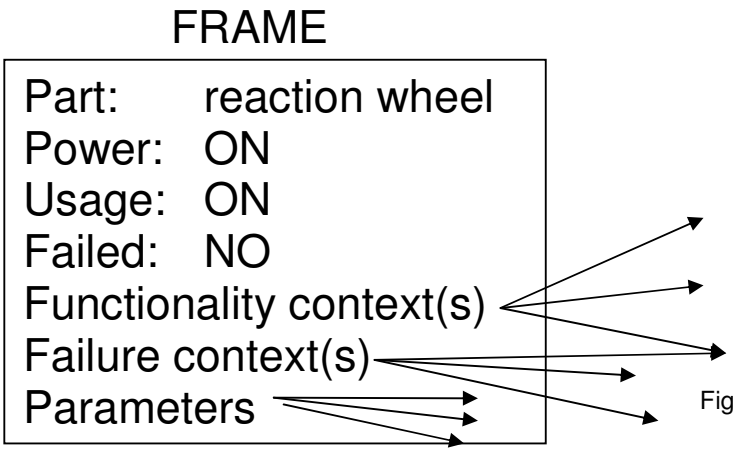


Fig 9.3 Common frame design

The frame (fig 9.3) of each device will be implemented as a table on the SQL server, maintained and changed by the updater and rules (with the influence of the operator). The Parameters item provides space for eventual extensions in the further overall system implementation, therefore will not be analysed in this work. Functionality and failure contexts are to be implemented as additional tables, providing nothing but logical addition or multiplication of other parts of the database, namely conditions of the failure/functionality and consequences of the failure/functionality. For example, a pressure sensor's functionality context requires the acknowledgement of the relevant power switch being on and delivering current (logical multiplication) and the consequent delivery of a measured value. For the value reliability check, also a cross-correlation with the relevant temperature sensors is made and if this proves the pressure sensor delivers non-reliable data, again, the functionality is questioned by changing the logical value of the final multiplication to 0. Unlike functionality context, the failure context is based mostly on logical addition, not demanding all the failure conditions to be fulfilled to raise an alert. Raising a parallel to the Intrusion detection systems described in chapter 5, this system is to be considered "paranoid". The entire cross correlation between the sensor readings is to be done by the updater filter.

- *Mission Model* – observed values changing boundaries dynamically, implemented as a combination of functionality frames and production system. This part is already partially reliably provided by the Mini Control System and OBSW, therefore it won't be part of this work's proof of concept prototype. However, for example the thermal testing and pre-calculated mission phase-related thermal behaviour check is a knowledge which should be incorporated in the overall finished product.

The inference **production rules** engine consists of:

- *Functional Model* – implemented in the production rules.

- *Operator Behaviour Check* – implemented by the combination of rules and database with the ability to undo the tested change. This functionality is already partially provided by the Pre-transmission validation tool.

The *ILP extension* called **Experience database** consists of:

A set of algorithmic descriptions of known possible extracurricular failures, both spacecraft and mission related. The yet undefined errors are to be reported by the operator in order to be automatically detectable by the second occurrence. The user interface input is crucial in creating such base.

The **Database updater and filter** consists of:

A filter of telemetry data, choosing the relevant items and translating them into a desired database form, another filter provides an alert handling for “out of limit” values, that is, a low pass filter of the received failure relevant data and interpolation to predict the probable fatal failure time of the part. Fortunately, a tool providing the mission-dependant, time-varying limits is available, therefore the filter simply compares such values. The updater will check every second (the highest frequency of realistic data download) and search for packet counter increase in every related observed table cell. The low pass filter will then be performed over a certain number of last values of the watched parameter, not over any given time period, as the frequency of data downlink might differ.

The ***user interface*** consists of:

An interface providing information on predicted and detected failures and recommended recovery procedures, as well as intended action feedback from the simulator, along with input possibility of intended operator actions and extracurricular failure database input.

The meaning of **mission planning** connection is:

A simple check can be implemented and run to compare the telemetry reported position and attitude to the planned one. However, as the mission planning tool is commonly not updated, the result of the check might lead rather to mission planner use enforcement than to a serious failure discovery. The use of mission planning tool with connection to an existing and available Sun vector tool might serve to improve the fine setting of limits for temperature ranges, leading to more sensitive anomaly detection. This part is only a suggestion for future improvement and will not be covered by the implementation.

9.2 Implementation

The implementation is surprisingly simple thanks to the structure of the database, which allows the implemented rules to be as simple as possible. The actual structure of programmed part consists mainly of filters (using a small database to be compared with another database), implemented as SQL queries, TC evaluators (again, calling a query to compare the TC with an existing one and another query to run a simulation of a given command), and DB checks, performed autonomously or upon request and again providing a DB query. The only two modules not working with the database in a straightforward manner are the one interfering with the user (GUI) and the one containing the simple production rules of detection.

As the structure will be mainly included in the SQL database and queries, a typical functionality and failure context is to be described first (continuing the frame description from fig. 9.3). A part of a typical context frame can be seen on fig. 9.4. Each context has two parts, conditions set and consequences set. The fields describing values derived from telemetry and from other fields will actually be cells of tables within a database. Obviously, a pressure sensor functionality cell is dependant on the entire functionality contexts of relevant temperature sensors, as the performance of the sensor depends heavily on its thermal comfort. Two contexts are therefore described by the figure at the same time.

The structure of the Experience database and the Knowledge base, as named on fig. 9.2, can be in the case of satellite FDIR connected into one database, assuming of course the possibility to distinguish between a failed part and a general failure case. As will be shown later, the tables implemented in the module structure of the system include item (part) understanding of the system as well as the event (case) approach, providing extendibility, thus being compliant to the Inductive Logic Programming theory described by Lavrač and Džeroski [20].

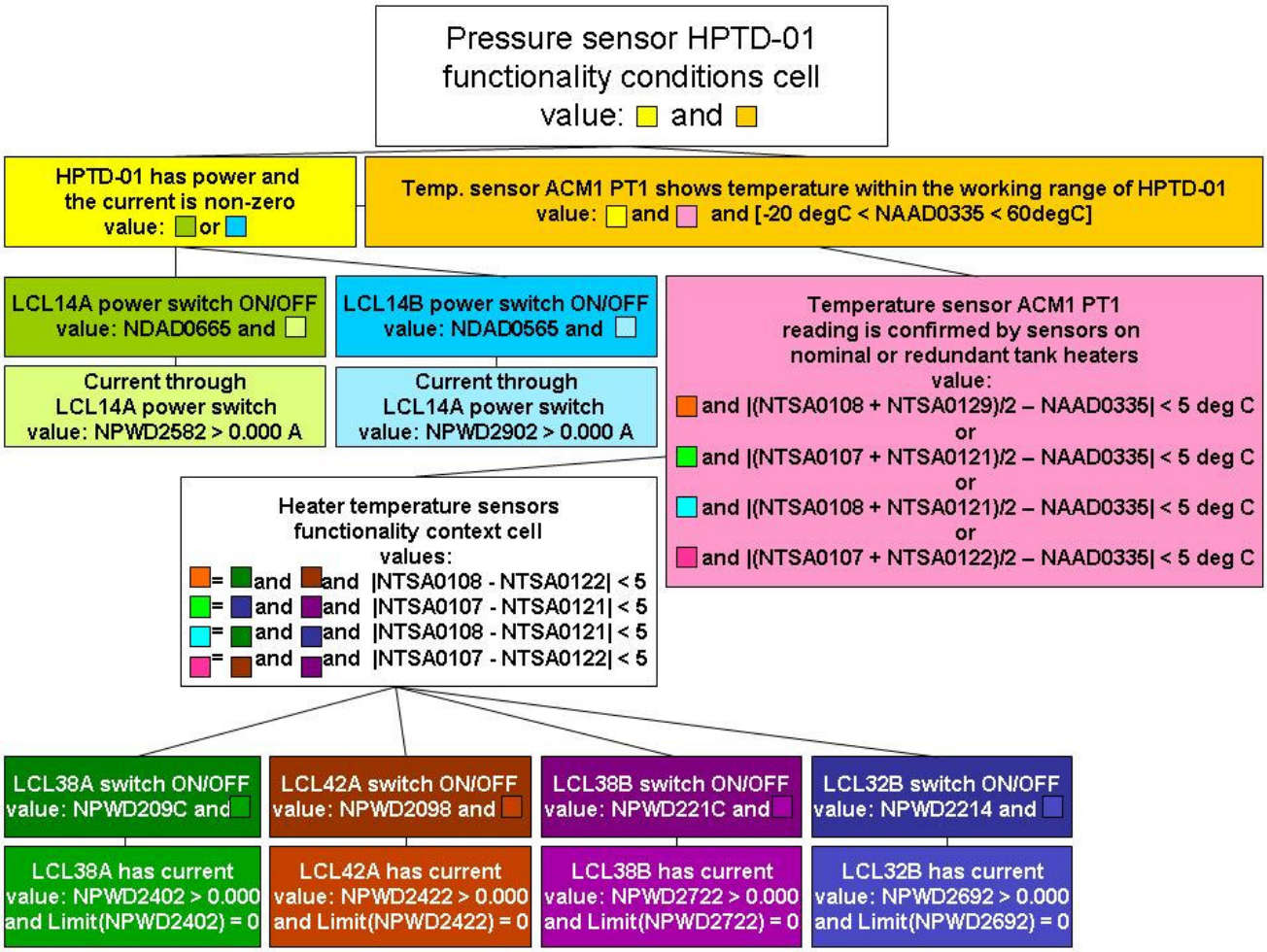


Fig 9.4 Pressure sensor functionality conditions example

The strict logical AND in most of the frames is typical for functionality context, as the functionality conditions and consequences have to be fulfilled all. To make the strictness of the schematic more obvious, fig. 9.5 provides a slightly less detailed flow chart of the same functionality conditions context. As can be seen, except of the redundant parts, every negative response leads to a non-functionality report.

On the other hand, a failure context provides several possible failure conditions, therefore a typical logical operation is OR. The formulation of both functionality and failure context might seem rather redundant, however, as explained in chapter 8, both of the definitions are needed for specific diagnostics.

It should be emphasized, that both functionality and failure contexts consist of conditions and consequences. However, as functionality contexts are simpler to derive (usually the device does or does not deliver a performance in expected values), the conditions contexts are more complex and therefore interesting for analysis.

Because most of the functionality contexts require temperature, power and sometimes pressure constraints, power and temperature sensors are considered the lowest level. Typical temperature sensor functionality condition is merely a power input. Typical power switch functionality condition is a current going through when the switch is on. As the target subsystem is the propulsion system, not power or thermal, the functionality check of related devices will be rather shallow compared to propulsion related devices.

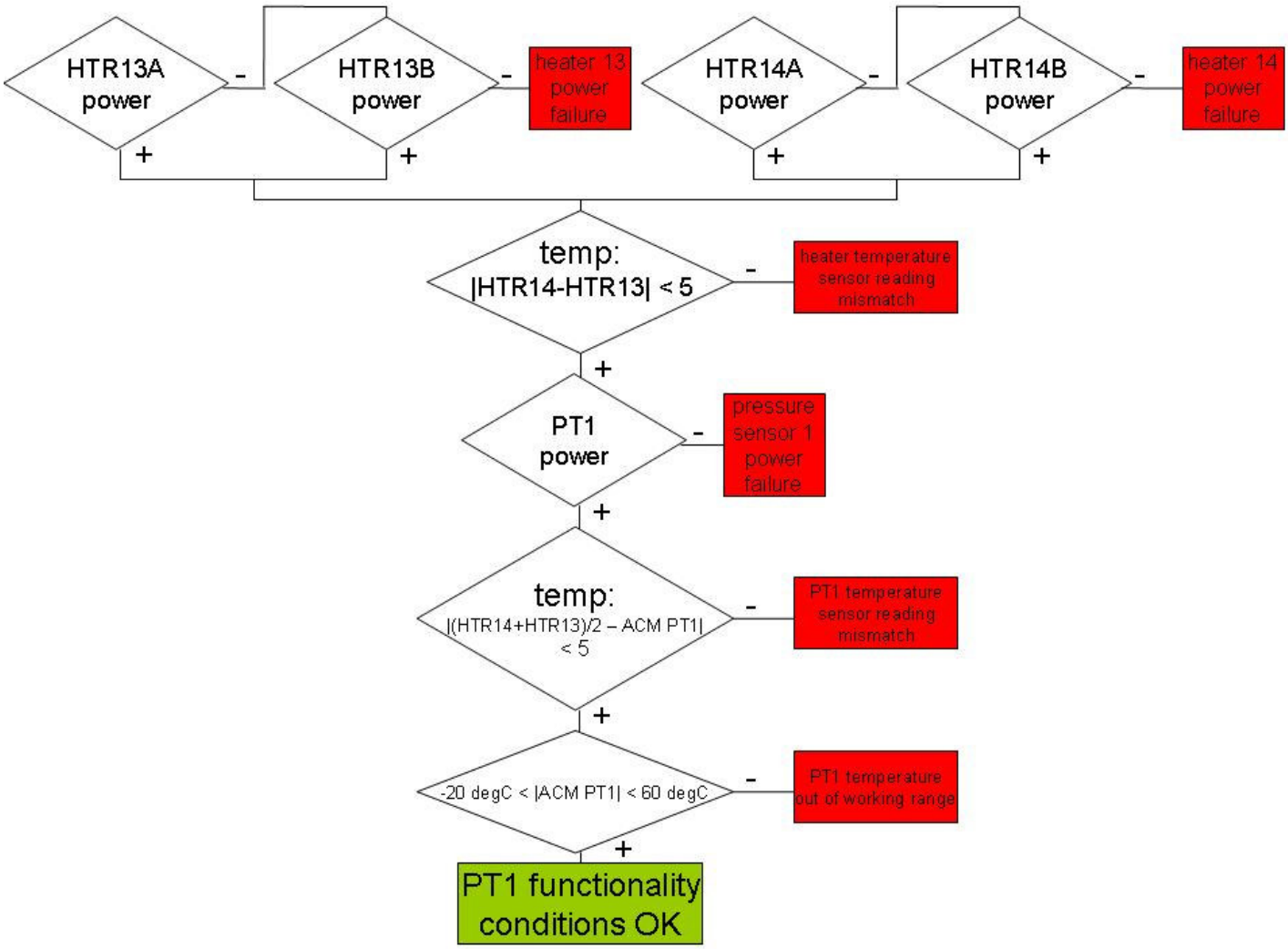


Fig 9.5 Pressure sensor functionality conditions flow chart

Unlike the database relations, the overall structure of the software is easy to depict and process using UML.

As visible from the detailed block scheme on fig. 9.6, not too different from 9.2, several modules will be created to communicate with each other, the user and the database. The communication with the satellite is simplified to an output that will be fed to simulator TC and input from TMCatcher database.

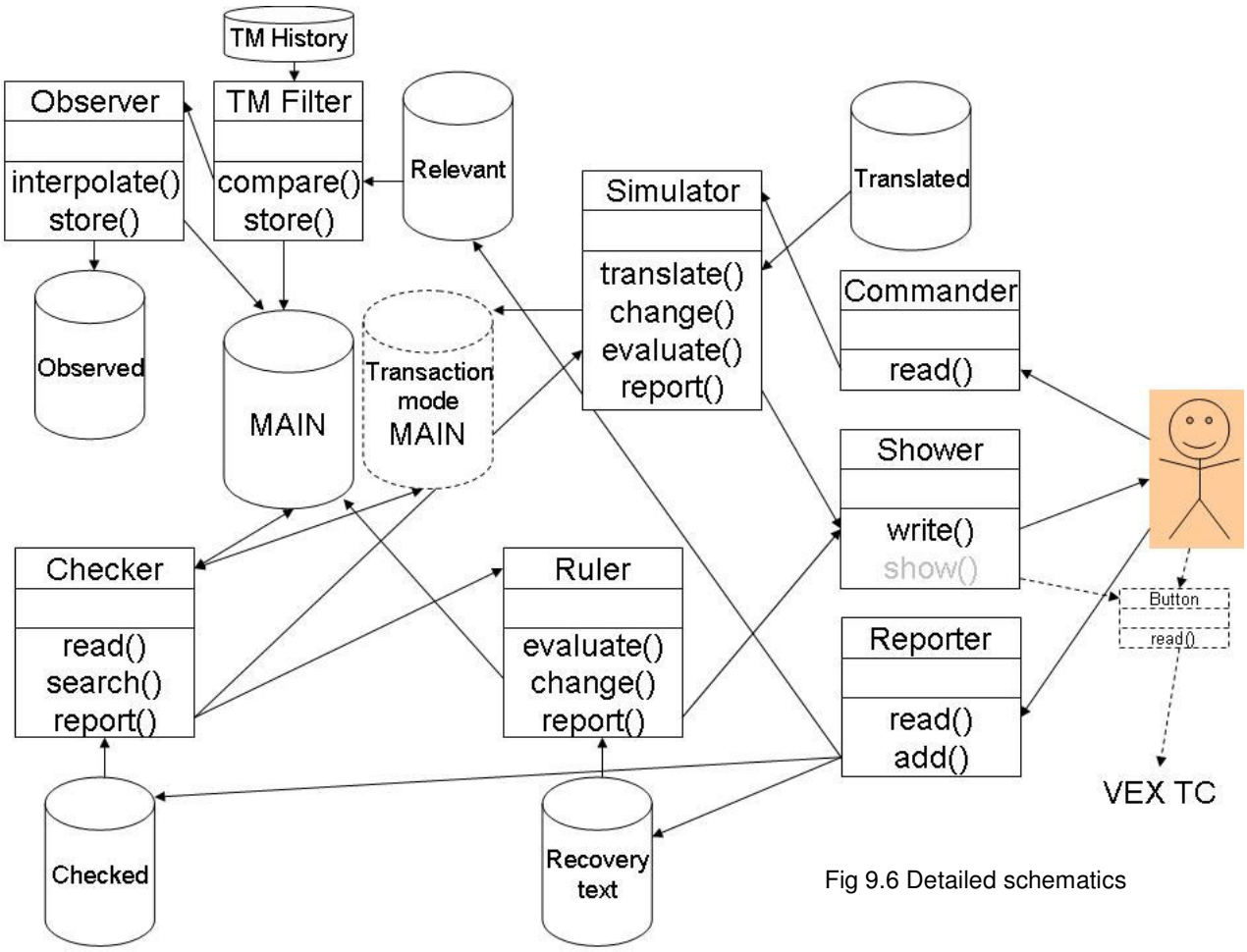


Fig 9.6 Detailed schematics

9.2.1 Modules details

TM Filter: A small, extendable database provides the list of parameters to be selected and copied from TMCatcher and TMHistory database. A TMHistory check is performed in a loop to detect an increase in a packet number at any relevant parameter. For fast search in the TM database, two simplified tables are dynamically created by queries, a table of relevant packet numbers (can be increased by the user extension) and the table of parameters being downloaded for the given packet number increase detected. The increased packet number parameters are updated into the Knowledge base into known frames. Any out of limit in relevant parameters triggers a call of Observer, for which another small working table is used as a trans-storage between the original database and the main knowledge base.

Observer: For any out of limit reported in TMCatcher, observer is called by the TM Filter. Observer stores a set of several last values of such parameters and provides an extrapolation, calculating the time left before the hard limit is reached from the trend of the value observed. If the out of limit state is over, the Observer keeps storing the values until the floating average reaches within limit state. This information is included to dedicated cells in the Main and Checker table.

Commander: The user input interface, providing the access to the simulator. Considering the user not capable of responsible behaviour, every command is first fed to the Simulator and feedback is provided, requiring the user to approve that this is the state of the system desired. First after the feedback is shown (by Shower), the user is theoretically allowed to click a “Really send” button.

Simulator: Called by the Commander, induces the DB “transaction mode” and remains in it until user approval. Performs a given command in the transaction mode and calls the Checker on the transaction protected database to obtain a feedback. If the feedback doesn’t show any

failure or danger, optionally this module can send the command without further user verification, however, by default, this option won't be implemented.

Checker: Either called by the Simulator or running in a loop, performs a check of the failure and functionality context relevant cells of the frames (which are grouped into an external table, which is actually queried by this module) and in case of an unwanted value performs a search among the grouped values to detect the failure/non-functionality source. This information is then used in a call of the Ruler. Checker passes all the out of limit time warnings to the Ruler.

Ruler: Ruler applies the rules, described in chapter 8, to detect the type of failure and evaluate. Then accesses the separate Recovery database to obtain the whole set of information and calls the Shower to pass this information to the user. All the out of limit countdown time warnings are passed to the Shower.

Shower: Called by the Ruler or by the Simulator, the Shower merely shows the detection results and when called by the Simulator, enables the Commander "Really send" button afterwards to allow user to validate the TC based on the feedback.

Reporter: Induced by the user, enables the expert knowledge extension. Physically, the Reporter adds cells to the Checker table, to the Ruler table (to verbally explain the recovery) and if necessary, to the TM Filter table. The Checker table extensions are filled with logical additions or multiplications of other known cells and the TM Filter table extensions are merely those parameters, that have not been downloaded from the TMCatcher database before, as previous knowledge did not consider them relevant. This property of the expert system is derived from the Inductive logic programming theory and enables even the broadening of this subsystem's model onto the whole satellite.

However, the overall satellite knowledge base would require an optimalisation of the computations and database organisation, therefore using the expert system "as is" and merely adding knowledge would not be wise. This prototype is only a proof of concept model and further broadening requires higher programming skills and computational capacities.

Obviously, six tables will be used in the database structure. First and the most important is the **main knowledge base**, including previously well defined frames, consisting of item name, ON/OFF information, power information, parameters and failure and functionality contexts.

This database is updated either by the Ruler (an item is detected to be failed is denoted as failed not to be used again) or by the TM Filter, which uses its own simplified, extendable database, including only relevant parameter names and their packet count numbers. Based on this, any change in TMCatcher database induces an update of the knowledge base. When extended, **TM Filter table** is informed about a new parameter to observe and simply has a cell added in order to update the main database with the defined parameter. However, such a parameter in the database would only be a conditional type of information, used in a failure or functionality context, not a whole new added item.

The following table of an insignificant size is the **Observer table**. Parameters incoming with an out of limit information are monitored by storing a set of several last TM values. The table will only consist of parameters and their maximum of 128 values. A tool should be provided to erase the old records once the sliding average of the value gets under the limit. The low pass filter will be designed less sensitive (more values involved) for the thrusters and the main engine temperatures.

The main database needs to be observed for failure and functionality context, which is optimalised by the **Checker table** existence. The Checker table is again only a simplification of the main database table, including only item name and functionality and failure context cells in order to be faster to read in a cyclic manner. Checker evaluates the values and if any of the contexts turns to the wrong value (functionality to 0 or failure to 1), it triggers the production rules application. The Checker table can be extended in an item-free manner as well, defining not a failed item line, but a general failure name and its context.

That leads to the utilisation of the last table in the process, the **Ruler table**, including again a table of possibly failed items (or generally failures) and cells with plain text description of recovery procedures. Specific recovery actions can thus be defined either when creating the database or by the user interface.

And as an extra, a rather different from the others, the **Simulator table** is incorporated, to provide known relevant TC and its effect on TM for simulation purposes. For example, TC that is not in the database is reported as invalid, TC leading to an increase of temperature in a part that is overheating raises a warning, etc. The structure of such table is simply telecommand and the names of parameters that are influenced along with the parameter influences, described in an algorithmic way.

As mentioned before, the tables are the part holding the knowledge, whereas the inference engine merely provides a cyclic comparison of obtained results. The inference engine, consisting of the Checker and the Ruler, is one of the two cyclically running processes. The other one, the database updater module, consisting of TMFilter and Observer, is no less of an importance. Access macros allow the run of two cyclic operations at the same time, however, the transaction mode of the MAIN database table can not be implemented, therefore the same functionality will be used within MCS (Mini Control System). An important note is, that for the prototype, MCS is used as a source of data instead of a satellite, therefore the Simulator part implementation might be implemented as a stub of interconnection to the MCS instead of re-programming the entire tool. However, the actual MCS simulator in principle complies with the one depicted on fig. 9.6.

9.2.2 Issues considered and encountered

During the implementation of the knowledge based expert system, shown in the Appendix B, several issues have been encountered. Firstly, the necessity of MS Access usage lead to unavailability of SQL transaction mode, which is the main essence of the simulator part. However, as the data, considered by the project to be incoming telemetry, are actually generated by a simulator much more advanced than the simple intended model, the simulator part was decided to be removed from the project as an unnecessary proof of already implemented concept. The implementation designed in this work is not entirely followed in the simulator, however, re-designing would be a matter of manual work, as the difference between the knowledge based approach and the actual state of the simulator software is merely a difference in denotation of used data in the database.

The implementation of knowledge based approach (algorithm should be simple and the knowledge should be stored in a database) does not allow using any knowledge to create the macros within Access. However, a table can carry an information on which query is supposed to be used for specific cases and various kinds of queries can be applied, using the fact that similar parts of the satellite provide similar behaviour.

Another problem encountered is the information redundancy. For example, a parameter providing information on power switch LCL14A is delivered independently in two telemetry packets. However, the redundancy of information is already handled by the MCS, therefore in the framework of this project, the redundant data will be considered unique.

Different redundancy in information evolves even within the constructed knowledge base. For example, a temperature sensor on a tank has some reading, which is important for both the tank status and the temperature sensor status. In case the temperature reaches an out-of-limit for a temperature sensor proper functionality, the temperature sensor should be denoted as failed. However, the tank has different temperature constraints and therefore its temperature information is handled in another way. Nevertheless, both of these two parameters, belonging to two different units, are actually the same value, coming as one parameter in one packet. The redundancy issue within the knowledge base might require further optimalisation from the performance point of view. However, from the point of view of this limited project, the redundant data are allowed to be stored, decreasing the performance, but providing more transparent information inside observed tables, which is an important

aspect for a pattern project to be enhanced and copied in the future. The trade-off between transparency and performance is easy to decide as the finished tool should preferably provide transparency.

For the same reason, many innuendos for extensions were implemented within the tables (for example COUNTSTEPS column, placed for the future use, meant to store the failure countdown value re-calculated into the number of times the operator is able to react). Also distinguishing between ON/OFF and IN_USE states might seem irrational, however, the extension is needed for other parts of the satellite, when an item can be turned on or off and used or not. The combination of various values of these two columns and the POWERED column enables the fine definition of redundancy types. The extensions are not meant to be performed on the very same tables and queries, however, certain inheritance from this project is to be expected.

It should be emphasized that the task of the model is not to provide a full knowledge of the selected subsystem, but merely show that with sufficient knowledge, such expert system would be a useful tool. The knowledge gathered about the subsystem is sufficient to provide simple comparisons based on gas law, limits knowledge and extrapolation, an expert using the delivered system is expected to provide more thorough and interconnected knowledge which might be out of knowledge range of the expert system programmer.

Another issue of the limited model is its incompleteness, leading to the need to take certain parameters for given, as verifying them would lead to further broadenings of the subsystem observed, until the majority of the satellite would be involved. For example the temperature readings are taken for granted as long as the Remote Terminal Unit is powered, as evaluating the quality of the temperature reading translation and transmission is far beyond the frame of this small prototype.

The environment to operate with is another field raising issues to be solved. As the final tool should provide even pre-transmission validation extension, the power to interfere with sent telecommands is necessary. However, the prototype is not given such privileges, as it is only meant to prove the ability to do so. Therefore the whole TC related function of the PreDetIR is to be omitted, left to be proven as easily possible by the similar functionality over TM.

Figure 9.7 provides detailed schematics of the fully functional tool, even broader than the overall descriptive design, whereas figure 9.8 shows the actual implementation for the prototype purposes.

The originally intended SQL implementation would provide the transaction mode possibility, however, the Access only enables the option to create a copy of a table and induce changes on it. That is a very poor and limited version of the same, leading to, as mentioned before, omitting the Simulator part, as it would be redundant to an existing simulator functioning on the very same principle within the MCS.

Inside the Observed table, an implementation of FIFO was added [27], handling the observed data and the low-pass filter in a very transparent way.

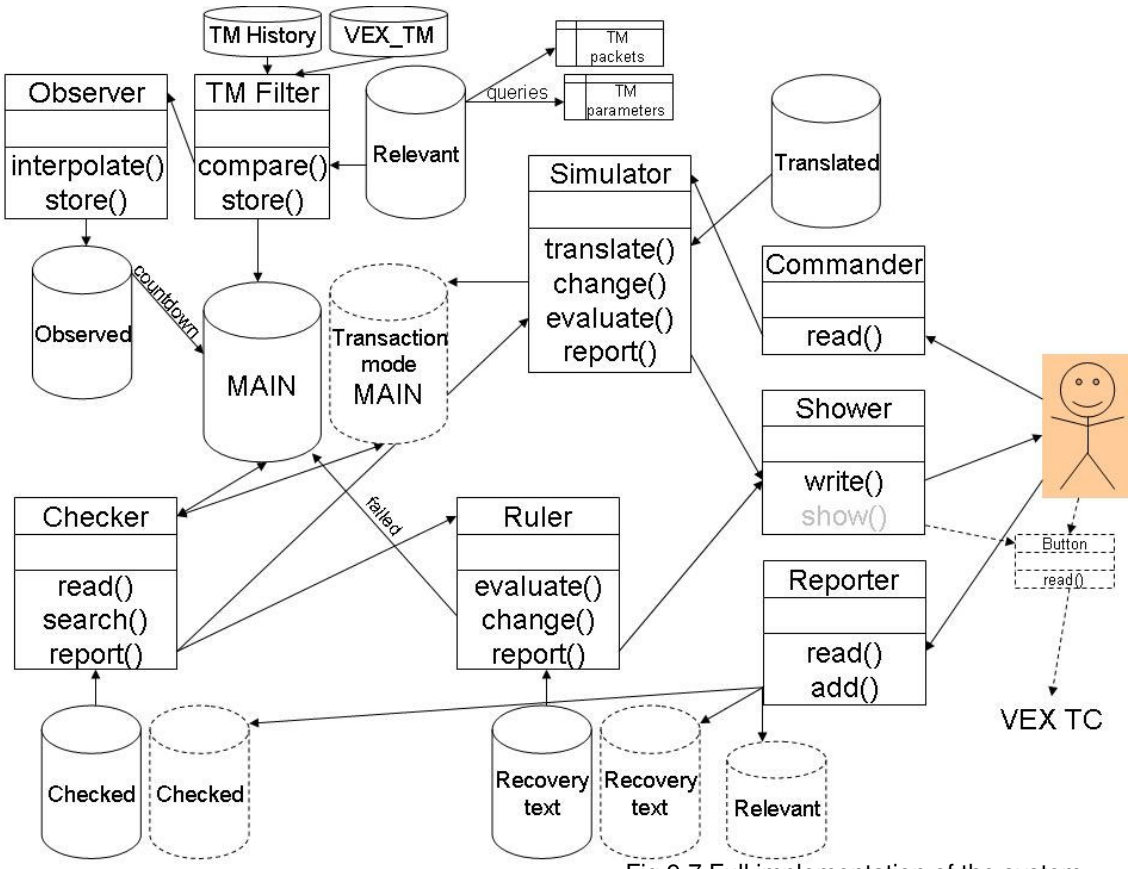


Fig 9.7 Full implementation of the system

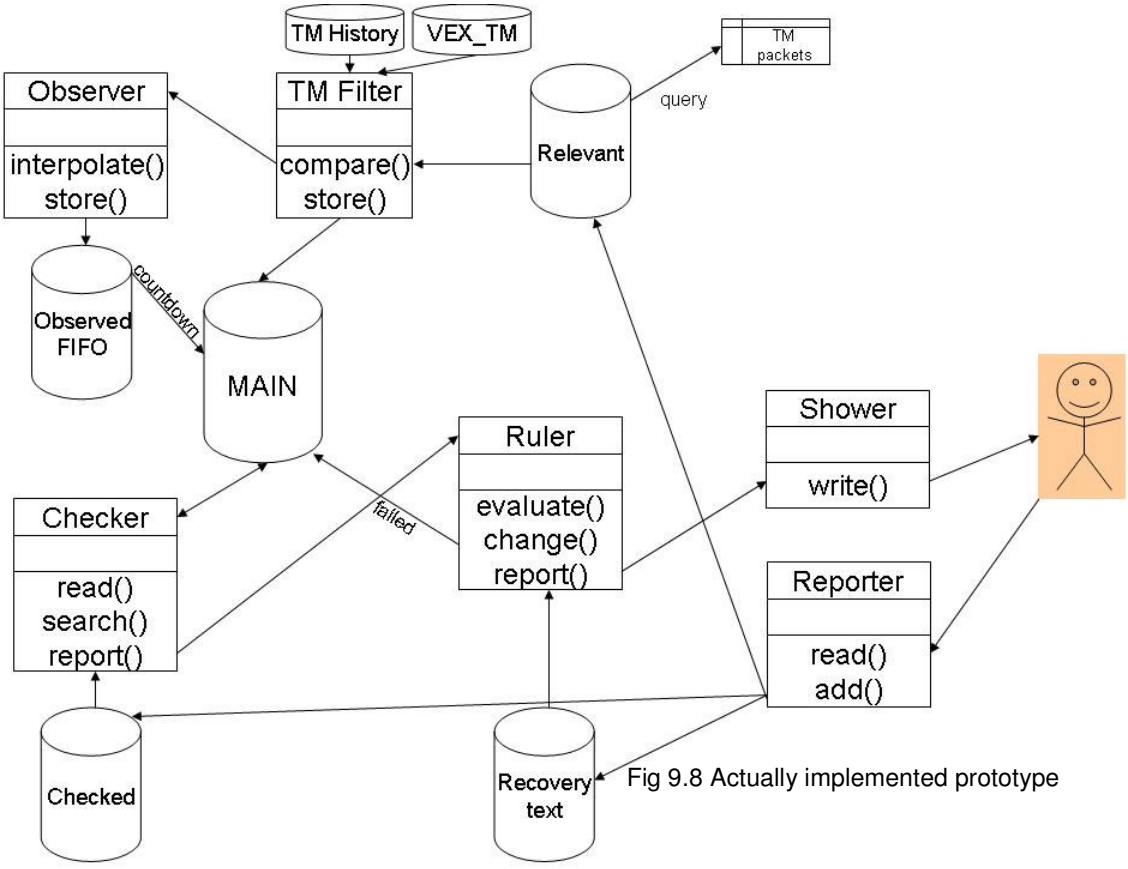


Fig 9.8 Actually implemented prototype

9.3 Debugging and testing

The testing stage, preceded by the debugging stage, had requirements on the staff of VEGA company, as the need of the satellite simulator data appeared. As the GUI (Graphical User Interface) of the tool has been developed and first results have been obtained (fig. 9.9), using quite simple situations results, new ideas for the failure interpretation methods and requirements appeared. The necessary equipment of such tool is a control panel, providing at least start and restart buttons (fig 9.9). Such part of the GUI has been developed and surprisingly provides the full access to all functionalities of the knowledge system.

Fig 9.9 Control panel

The detected failures are notifying the user by a pop-up window (fig. 9.10), an annoying, but important tool, forcing the operator's immediate reaction. The decision is to be made whether to display the failure window for all kinds of failures (potential, functional, process and hardware) or just for these requiring the immediate reaction (hardware and functional).

Fig 9.10 Pop-up window

After individual notifications, all reported failures are stored in a history log table along with their timestamps and displayed by the form (fig 9.12). Countdown information is naturally included in both cases, as long as the countdown is relevant.

All the mentioned parts are fully functional and deliver failure prediction, detection, identification and recovery hints for all the cases known to the database creator.

However, the entire expert system is not just a tool for the operator, the major functionality is given by the interface allowing the user to enhance the knowledge by extending either the simple knowledge base or the experience base. These two options are virtually identical, only the experience data allow the user to define the knowledge and principles in an algorithmic way, using the string handling power of nowadays programming tools.

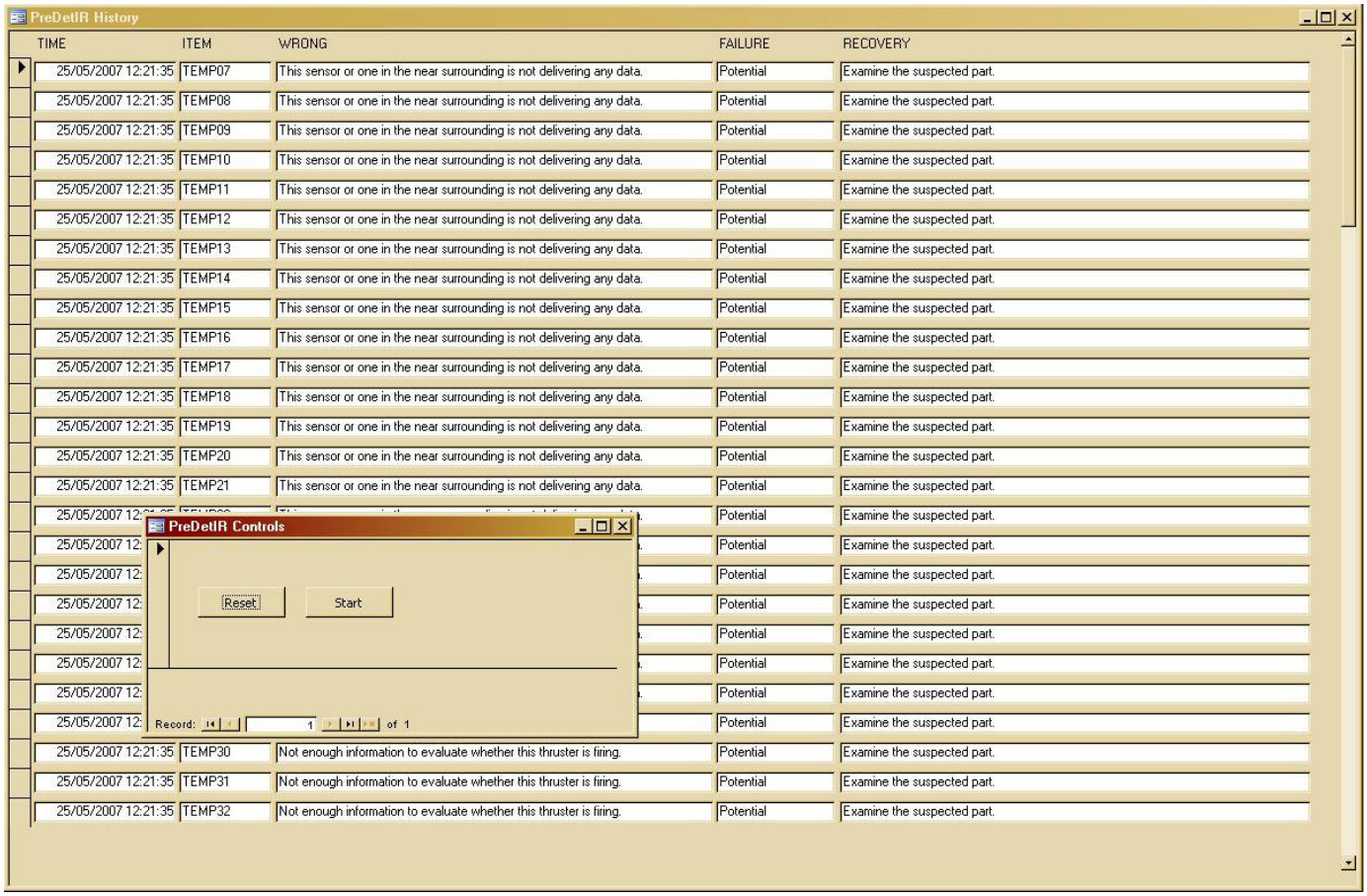


Fig 9.11 GUI with the first results

The testing of the tool has been performed in the offices of VEGA on 25. 5. 2007. First random packetstream of data didn't contain all the information required by the system, leading to series of warning messages. Therefore a method of testing has been chosen to prevent such floods of messages (fig 9.11), which were causing difficulties in the real testing and debugging process.

A set of nominal behaviour data was created by the simulator and introduced to the system. Afterwards, several sets of data were generated by the simulator, showing faulty behaviour of chosen parts or their combinations. The output of the system has shown as a success (especially different limit settings for interconnected parts, allowing the tool to distinguish between the real part failure and the failure of a housekeeping system), though more thorough information on the failures and their background was pointed out as an issue, as the formulations provided by the knowledge base were somewhat vague. However, the PreDetIR system has proven its full functionality, thus proving the concept of the knowledge based expert system as applicable.

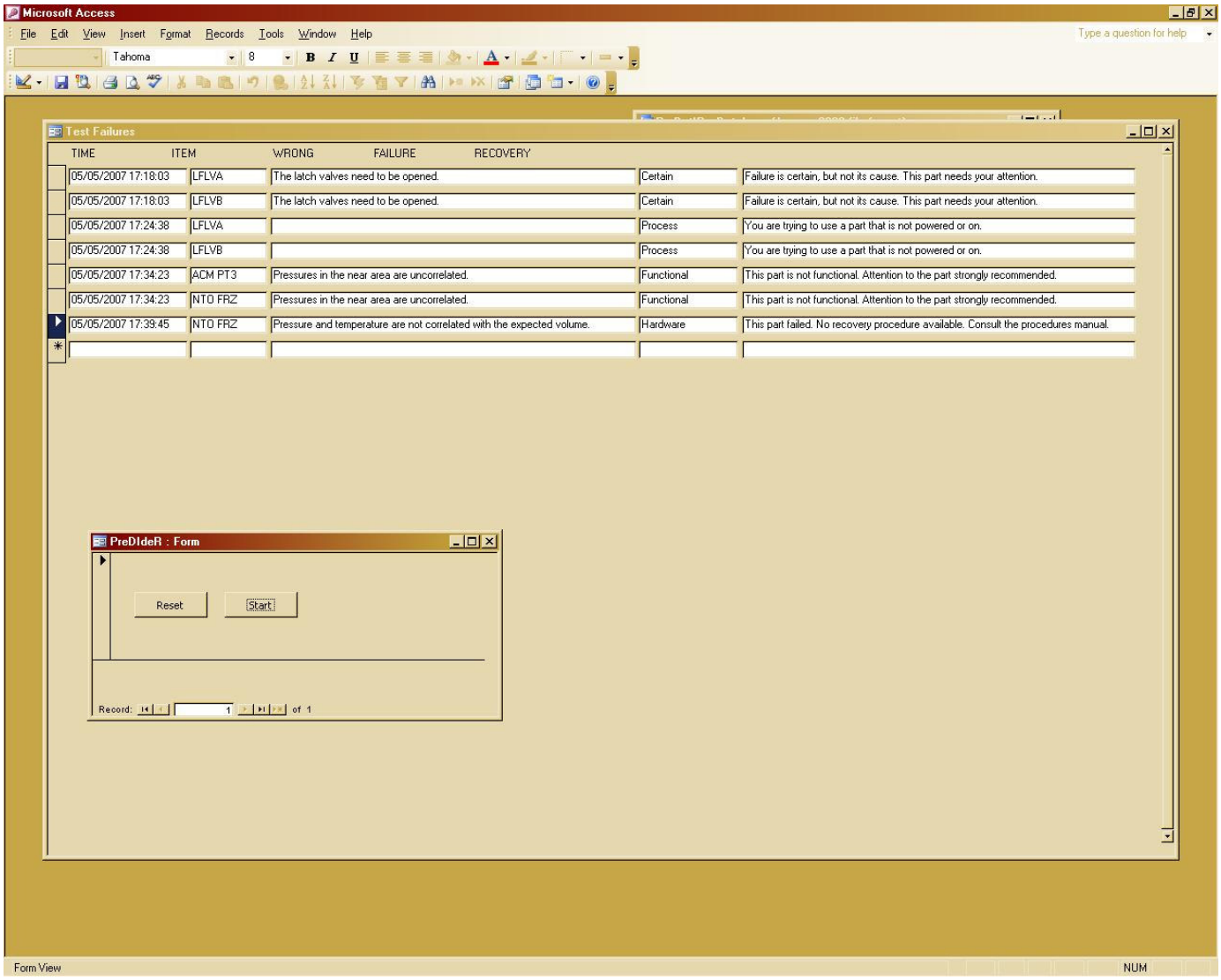


Fig 9.12 GUI with performance proving results

Another part of the knowledge based system is a learning mechanism. This has been implemented as a form, where a human expert can type or choose items to add knowledge about. Specifically, the relevant and main tables are updated if the expert claims more items to be needed to be observed, the checked table and recovery table are broadened by the knowledge on how to recognise a failure within the observed part of the system. The entire user interface is implemented as one form window with an intuitive communication (fig. 9.13).

To access this editing form, extra control has been added to the main window. Also an extra button for a self-standing check without a download of new data has been implemented for repeated experiments (fig. 9.14).

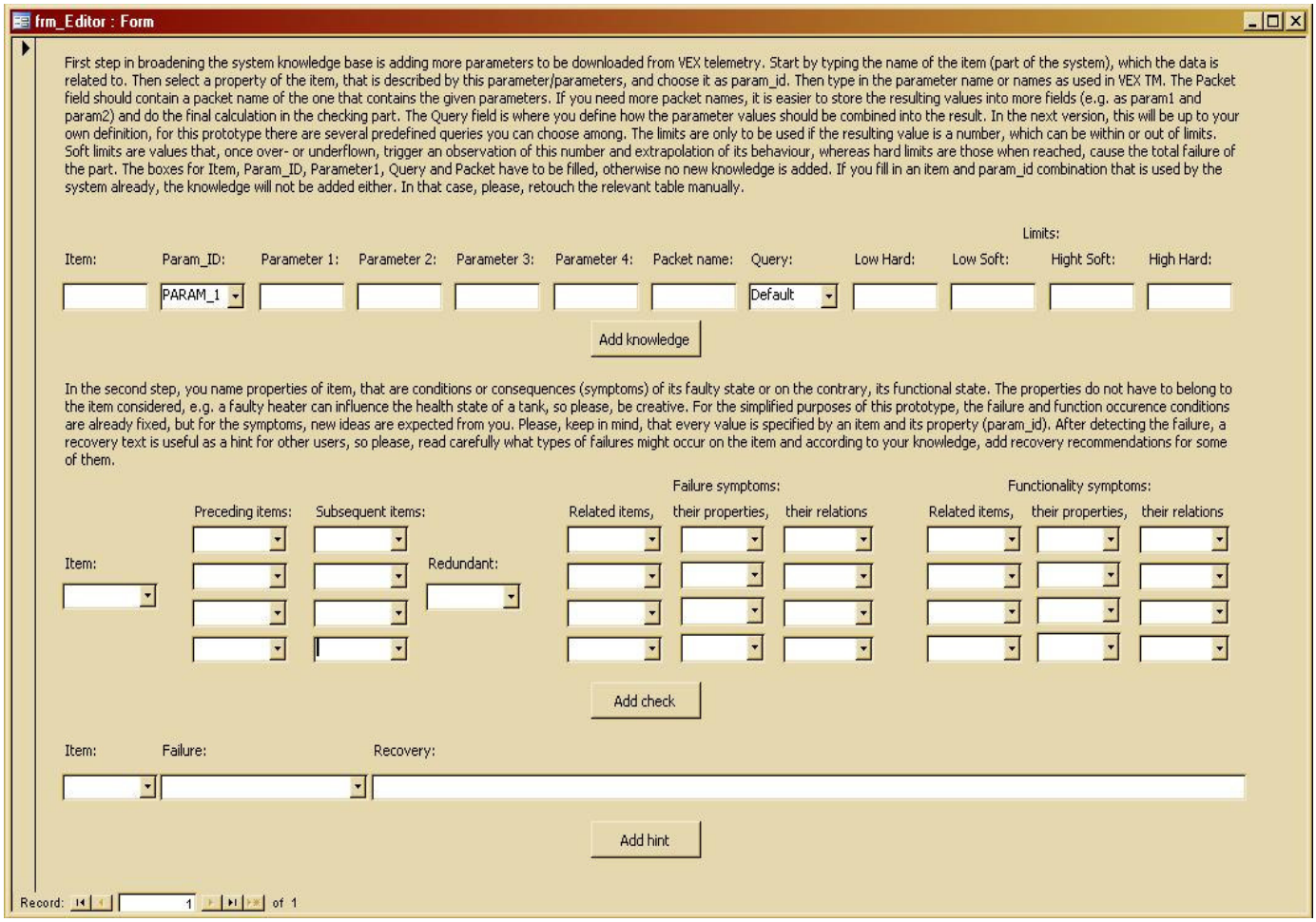


Fig 9.13 Editing user interface

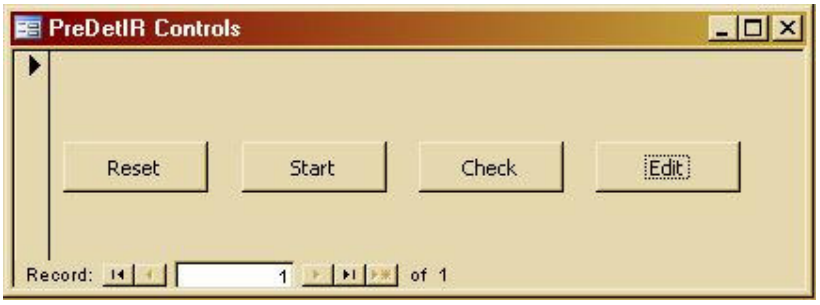


Fig 9.14 Final control panel

10 Conclusion

An evaluation of the available FDIR methods has been performed (Chapter 4) and the optimal strategy for on-ground satellite operator tool has been chosen (Chapter 6). A simple proof of concept prototype has been developed (Chapter 9), providing all the functionalities of the chosen knowledge based, inductive logic programming extended solution (Appendix B). Desired performance has been tested, using the Venus Express simulator (Chapter 7) and positive results have been reported.

The tool has been implemented using MS Access and Visual Basic for Applications (VBA), following in detail the requirements of the knowledge base approach, as well as inductive logic programming.

In case of the whole system implementation, the resulting tool would support the mission control operator with fast failure prediction, detection, identification and recovery suggestions.

The MS Access tool has proven itself to be powerful and suitable, though several functionalities of the designed system must have been omitted or limited. The option of using a GUI (Graphical User Interface) made the entire tool very valuable for future usage by non-computer background satellite system operators.

However, a thought occurs that the operator's comfort and aid might cause a degradation of the future satellite operator's qualities, thus lowering the overall requirements on human education and understanding of the world and the tools used for its observation. Decisions have to be made by the authorities, whether such approach is the desired future of the mankind.

11 References

[1] Frank, P.M., *Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge-based Redundancy – A Survey and some results*, Automatica, Vol. 26, No. 3, pp. 459-474, 1990

[2] Doraiswami, R., Diduch, C.P., Kuehner, J., *Failure Detection and Isolation: a New Paradigm*, Proceedings of the American Control Conference, Arlington, VA, 2001

[3] Stergiou, Ch., Siganos, D., *NEURAL NETWORKS*, http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[4] Wikipedia contributors, *Neural network*, Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Neural_network&oldid=114578959, 2007

[5] Debar, H., *What is knowledge-based intrusion detection?*, Intrusion Detection FAQ, http://www.sans.org/resources/idfaq/knowledge_based.php, 2003

[6] Debar, H., *What is behaviour-based intrusion detection?*, Intrusion Detection FAQ, http://www.sans.org/resources/idfaq/behaviour_based.php, 2003

[7] Halme, L.R., Bauer, R.K., *AINT Misbehaving: A Taxonomy of Anti-Intrusion Techniques,* Intrusion Detection FAQ, http://www.sans.org/resources/idfaq, 2003

[8] Farshchi, J., *Statistical based approach to Intrusion Detection,* Intrusion Detection FAQ, http://www.sans.org/resources/idfaq, 2003

[9] Sarle, W.S., *Neural Network FAQ, part 1 of 7: Introduction, periodic posting to the Usenet newsgroup comp.ai.neural-nets*, URL: ftp://ftp.sas.com/pub/neural/FAQ.html, 1997

[10] Klein, M., Dellarocas, Ch., *A Knowledge-based Approach to Handling Exceptions in Workflow Systems*, Computer Supported Cooperative Work 9: 399-412, 2000

[11] Wikipedia contributors, *Probably approximately correct learning*, Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Probably_approximately_correct_learning&oldid=85843259, 2006

[12] Briggs, K., *Valiant's theory of the learnable,* http://keithbriggs.info/documents/learnable.pdf, BTexact, 2003

[13] Unpublished, Christian Bodemann

[14] Muggleton, S.H., *Inductive Logic Programming*, http://www.doc.ic.ac.uk/~shm/ilp.html, London, 2007

[15] Valiant, L.G., *A Theory of Learnable*, Communication of ACM, Volume 27, Number 11, 1984

[16] Mattos, N.M., *An Approach To Knowledge Base Management*, Springer-Verlag, 1991

[17] Chiang, L.H., Russell, E.L., Braatz, R.D., *Fault detection and diagnosis in industrial systems*, Springer 2001

[18] Bodemann, C.D., *Operational Simulator Venus Express,* Vega GmbH, 2006

[19] Cannady, J., *Artificial Neural Networks for Misuse Detection*, Proceedings of the 1998 National Information Systems Security Conference (NISSC'98), Arlington, VA, 1998

[20] Lavrač, N., Džeroski, S., *Inductive Logic Programming – Techniques and Applications*, Ellis Horwood, New York, 1994

[21] Haykin, S., Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice Hall, New Jersey, 1999

[22] EUROPEAN COOPERATION FOR SPACE STANDARDIZATION, *Space product assurance, Failure modes, effects and criticality analysis (FMECA)*, ESA Publications Division, Noordwijk, The Netherlands, 2001

[23] Lavagna, M., Sangiovanni, G., Da Costa, A., *Modelization, Failures Identification and High-level Recovery in Fast Varying Non-linear Dynamical Systems for Space Autonomy*, Dynamics and Control of Systems and Structures in Space (DCSSS), 6th Conference, Riomaggiore, Italy, 2004

[24] Bade, A., *Columbus System Operations Concept*, Astrium Space Infrastructure, Bremen, Germany, 2002

[25] Johansson, A., Bask, M., Norlander, T., *Dynamic threshold generators for robust fault detection in linear systems with parameter uncertainty*, Automatica, Vol. 42 (7) pp. 1095-1106

[26] European Space Agency, *Venus Express*, http://sci.esa.int/science-e/www/area/index.cfm?fareaid=64, 2007

[27] Chris Rae, *Visual Basic for Applications (VBA) Pages*, http://chrisrae.com/vba/routines.html, 2001

[28] Patton, R.J., Frank, P.M., Clark, R.N., Fault Diagnosis in Dynamic Systems, Theory and Applications, Prentice hall, 1989

Appendix A: Telecommand and telemetry considered

The propulsion system uses the following telecommands:
Open LFLV-01
Close LFLV-01
Open LFLV-02
Close LFLV-02
Fire PVNC-01 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-02 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-03 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-04 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-05 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-06 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-07 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-08 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-09 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-10 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-11 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-12 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-13 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNC-14 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNO-15 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC

Fire PVNO-16 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Fire PVNO-17 Prime & Redundant and associated pre-arming, arming and dis-arming of pyro firing circuits/TC
Open TLV1A
Close TLV1A
Open TLV1B
Close TLV1B
Open TLV2A
Close TLV2A
Open TLV2B
Close TLV2B
Open TLV3A
Close TLV3A
Open TLV3B
Close TLV3B
Open TLV4A
Close TLV4A
Open TLV4B
Close TLV4B
Fire RCT-1A
Fire RCT-1B
Fire RCT-2A

Fire RCT-2B
Fire RCT-3A
Fire RCT-3B
Fire RCT-4A
Fire RCT-4B
Fire Main Engine (400N-ME) Prime
Fire Main Engine (400N-ME) Redundant

The propulsion system requires the following signals to be telemetered based on unit supplier provided equipment:
HPTD-01 Pressurant tank pressure
LPTD-02 Regulator outlet pressure
LPTD-03 Oxidiser tank pressure
LPTD-04 Fuel tank pressure
HPTD-01 Temperature
LPTD-02 Temperature
LPTD-03 Temperature
LPTD-04 Temperature
LFLV-01 Status
LFLV-02 Status
TLV1A Status
TLV1B Status
TLV2A Status
TLV2B Status
TLV3A Status
TLV3B Status
TLV4A Status
TLV4B Status
RCT-1A Chamber temperature
RCT-1B Chamber temperature
RCT-2A Chamber temperature
RCT-2B Chamber temperature
RCT-3A Chamber temperature
RCT-3B Chamber temperature
RCT-4A Chamber temperature
RCT-4B Chamber temperature
400N-ME Chamber temperature

In addition to the above the following propulsion system temperatures are also monitored using sensors provided by the thermal control system:
MMH tank top - nominal and redundant
MMH tank bottom - nominal and redundant
NTO tank top - nominal and redundant
NTO tank bottom - nominal and redundant
Helium tank - nominal and redundant
Pressure regulator outlet - nominal and redundant
LFLV 01
LFLV 02
CPS line 1 - nominal and redundant
CPS line 2 - nominal and redundant
CPS line 3 - nominal and redundant
CPS line 4 - nominal and redundant
CPS line 5 - nominal and redundant
CPS line 6 - nominal and redundant
RCT 1A valve
RCT 2A valve
RCT 3A valve
RCT 4A valve
RCT 1B valve
RCT 2B valve

RCT 3B valve
RCT 4B valve
ME MMH valve
ME NTO valve
ME flange (low)
ME flange (high)

Appendix B: The source code

```
Option Compare Database
 Dim X1 As Variant 'parameter values
 Dim X2 As Variant
 Dim X3 As Variant
 Dim X4 As Variant
 Dim X5 As Variant
 Dim X6 As Variant
 Dim PRECEDING1 As String
 Dim SUBSEQUENT1 As String
 Dim PRECEDING2 As String
 Dim SUBSEQUENT2 As String
 Dim PRECEDING3 As String
 Dim SUBSEQUENT3 As String
 Dim PRECEDING4 As String
 Dim SUBSEQUENT4 As String
 Dim item As String
 Dim param_id As String 'unique address of a record in all the tables
 Dim whatswrong As String 'for failure identification
 Dim NoInfor As Boolean
 Dim TIME As String
 Dim db As Database
 Dim rs_relevant As Recordset 'lookup table of items and their relevant parameters
 'Dim rs_main As DAO.Recordset
 Dim rs_main As Recordset     'table of system knowledge
 Dim rs_qselrel As Recordset  'result of qry_first, a selestion of the relevant table
 Dim rs_observed As Recordset 'table of observed out-of-limits values
 Dim rs_recovered As Recordset 'table of recovery hints
 Dim rs_VEX_TM As Recordset     'table of the fresh TM data
 Dim rs_checked As Recordset    'checking lookup table
 Dim rs_records As Recordset    'table of history log
 'for FIFO purposes
 Dim topix As Integer ' Where the NEXT thing goes on top
 Dim bumix As Integer ' Where the CURRENT thing is at the bottom

Private Sub Form_Load()
  Form_Timer
End Sub
```

```
Private Sub Form_Timer()
Main
End Sub
```

```
'the main sub, called periodically by the Form_Timer, calls the qry_first query, which compares packetnumbers saved in this DB with these saved in VEX_TmPacketDescription
'once difference is detected, the query generates a selection from the relevant table, including packet relevant names of parameters and their addresses in the main table
'the placement inside the main table is uniquely described by item name and param_id
'the values of the parameters are read out of VEX_TM and updated into the main table
'after updating the value, this is compared to its upper and lower limits and in case of out-of-limits being detected in the last or recent data, the Observer procedure is started
'after comparing the data, the previously changed packetnumber is updated from VEX_TmPacketHistory into the relevant table
'the Checker is called after the update to perform a failure detection in the new main table
'the functions Seeker and Searcher are providing the record number of the record addressed by item and param_id in observed and selected relevant table, respectively

Private Sub Main()
Dim rs_qtranTM As Recordset
Dim sqlString As String
Dim something As Variant
Dim poser As Integer
Dim j As Variant
Dim numerouno As String
Dim Critter As String
 'definitions
 Set db = CurrentDb
 Set rs_relevant = db.OpenRecordset("qry_relevant")
 Set rs_main = db.OpenRecordset("qry_main")
 Set rs_observed = db.OpenRecordset("qry_observed")
 Set rs_recovered = db.OpenRecordset("qry_recovery")
 Set rs_VEX_TM = db.OpenRecordset("qry_VEX_TM")
 'qry_first is a query selecting part of relevant table that is related to the newly incoming packet
 Set rs_qselrel = db.OpenRecordset("qry_first")
 'qry_second is a query that takes TMPacketDescription table and translates its received tim into a time-like value
 Set rs_qtranTM = db.OpenRecordset("qry_second")
 Dim n As Double
 'if the qry_first detects a new incoming packet
 If rs_qselrel.RecordCount > 0 Then
  'Call frm_TestFailures
 DoCmd.Close acForm, "frm_TestFailures"
 DoCmd.OpenForm "frm_TestFailures"
  rs_qselrel.MoveFirst
  PACKET = rs_qselrel!PACKET1
```

```
  With rs_qtranTM
   .MoveFirst
   .FindFirst ("[PACKET] ='" & PACKET & "'")
   If Not IsNull(!ReceivedTime) Then
                            TIME = CDate(!ReceivedTime)
                            Else:
                            TIME = "01/01/2000 09:25:27"
                            End If
  End With

'cycle on the selection of relevant table
rs_qselrel.MoveFirst
 Do
'HOW DO I KNOW WHICH ITEM???
  item = rs_qselrel!item
  param_id = rs_qselrel!param_id
'looking up the values of the parameters inside VEX_TM
  Call Looker(item, param_id)
poser = Positioner(param_id)
 'updating the main with new values from VEX_TM
 rs_main.FindFirst ("[ITEM] ='" & item & "'")
something = VALEX(item, param_id)
something = Trim(something)
If (poser < 4) And (poser > 0) Then
                                  If IsNull(something) Then
                                                       j = False
                                                       Else:
                                                       j = something
                                                       End If
                                Else:
                                j = something
                                End If
 If j = rs_main.Fields(poser).Value Then
                                  j = 0
                                  Else
                                  With rs_main
                                  .Edit
                                  .Fields(poser).Value = j
                                  .Update
                                  End With
                                  End If
With rs_qselrel
Critter = "[NO] =" & Searcher(item, param_id)
```

```
.FindFirst (Critter) '(!item = item) And (!param_id = param_id))
 'value is compared to the limits
 If Not IsNull(!HS) Then
If (CDbl(something) < !LS) Or (CDbl(something) > !HS) Then
                     Call Observer(item, param_id, !LH, !LS, !HS, !HH, CDbl(something))
                          Else:
 'if the value is in the limits, but the average still isn't, the observation continues
                           If (Seeker(item, param_id) <> 0) Then
                           If (CDbl(Average(item, param_id)) < !LS Or
CDbl(Average(item, param_id)) > !HS) Then Call Observer(item, param_id, !LH, !LS, !HS,
!HH, CDbl(something))


'if the value is in the limits and the average is in the limits, the record is emptied
in observed table
                                          If (CDbl(Average(item, param_id)) > !LS And
CDbl(Average(item, param_id)) < !HS) Then Call EmptyQ(item, param_id)
                                                                 End If
                           End If
 End If
 End With
 'if the value is in limits, but recently has been out of limits, Observer is called
anyway
 'updating the packet counter
 'input: VEX_TmpacketDescription
 'output: relevant
 With rs_relevant
 .FindFirst (Critter)
  .Edit
 !LAST_PACKETNR = DLookup("[Counter]", "qry_second", "[PACKET] ='" & !PACKET1 & "'")
 .Update
 End With
  'qry_update_first = "UPDATE relevant " & _
  '"INNER JOIN VEX_TmpacketDescription " & _
  '"ON relevant.PACKET1 = VEX_TmpacketDescription.NAME " & _
  '"SET relevant.LAST_PACKETNR = [VEX_TmpacketDescription].[Counter] " & _
  '"WHERE (((relevant.LAST_PACKETNR)<>[VEX_TmpacketDescription].[Counter]))"
  'DoCmd.RunSQL qry_update_first, 0
 rs_qselrel.FindFirst (Critter)
 rs_qselrel.MoveNext
 Loop While (Not rs_qselrel.EOF)

 Call Checker

 End If
```

```
rs_qselrel.Close
rs_main.Close
rs_relevant.Close
End Sub
```

```
Public Function Searcher(item, param_id) As Long
'returns the number of record where item = item and param_id = param_id in the relevant
selection, same as Seeker
'input: global variables item and param_id as Strings
'output: record number as integer
With rs_qselrel
Dim n As Long
n = 0
.MoveFirst
.FindFirst ("[ITEM] ='" & item & "'")
Do
  If (!param_id = param_id) Then
                                    n = !NO
                                    Exit Do
                               Else:
                                    .FindNext ("[ITEM] ='" & item & "'")
                                    End If
Loop Until (.EOF)
End With
Searcher = n
End Function
```

```
Public Sub Looker(item, param_id)
 'reading the relevant parameter names (String) from the relevant table generated lookup
table
 'looking up the values of the parameters in VEX_TM
 'input: table readings
 'output: values X1 to X6 as Variant, parameters of a given item
With rs_qselrel
.FindFirst ("[NO] =" & Searcher(item, param_id))
X1 = DLookup("[VALUE]", "qry_VEX_TM", "[PARAMETER] ='" & !parameter1 & "'")
If X1 = "ON state" Or X1 = "Active" Then X1 = True
If X1 = "Inactive" Or X1 = "OFF state" Then X1 = False
If Not IsNull(!parameter2) Then
                            X2 = DLookup("[VALUE]", "qry_VEX_TM", "[PARAMETER] ='" &
!parameter2 & "'")
                            If X2 = "ON state" Or X2 = "Active" Then X2 = True
                            If X2 = "Inactive" Or X2 = "OFF state" Then X2 = False
                            End If
```

```
If Not IsNull(!parameter3) Then
                            X3 = DLookup("[VALUE]", "qry_VEX_TM", "[PARAMETER] ='" &
!parameter3 & "'")
                            If X3 = "ON state" Or X3 = "Active" Then X3 = True
                            If X3 = "Inactive" Or X3 = "OFF state" Then X3 = False
                            End If
If Not IsNull(!parameter4) Then
                            X4 = DLookup("[VALUE]", "qry_VEX_TM", "[PARAMETER] ='" &
!parameter4 & "'")
                            If X4 = "ON state" Or X4 = "Active" Then X4 = True
                            If X4 = "Inactive" Or X4 = "OFF state" Then X4 = False
                            End If
'If !parameter5 <> Null Then X5 = DLookup("[VALUE]", "qry_VEX_TM", "[PARAMETER] ='" &
!parameter5 & "'")
'If !parameter6 <> Null Then X6 = DLookup("[VALUE]", "qry_VEX_TM", "[PARAMETER] ='" &
!parameter6 & "'")
End With
End Sub
```

```
Public Function Positioner(param_id) As Integer
Dim n As Integer
'holding the information on which column in main has which number
'input: global variablr param_id as string
'output: column number as integer
Select Case param_id
Case "POWERED"
        n = 1
Case "ON_OFF"
        n = 2
Case "IN_USE"
        n = 3
Case "PARAM_1"
        n = 7
Case "PARAM_2"
        n = 8
Case "PARAM_3"
        n = 9
Case "PARAM_4"
        n = 10
Case "FAILED"
        n = 4
Case "COUNTSTEPS"
        n = 6
Case "PARAM_5"
        n = 11
```

```
Case "PARAM_6"
        n = 12
Case "PARAM_7"
        n = 13
Case "PARAM_8"
        n = 14
Case "PARAM_9"
        n = 15
        End Select
Positioner = n
End Function
```

```
Public Function VALEX(item, param_id) As Variant
'handling the parameter values in known queries
'input: relevant table selection, X1 to X6
'output: VALEX As Variant, calculated value of an item parameter
 Dim X As Variant
 Dim T As Double
 Dim P As Double
 Dim V As Double
 Dim M As Boolean
  With rs_qselrel
 .FindFirst ("[NO] =" & Searcher(item, param_id))
 Query = !Query
 End With
 Select Case Query
 Case "Default"
        X = X1
 Case "Power"
        X = X1 And (CDbl(X2) > 0#) Or X3 And (CDbl(X4) > 0#)
 Case "Power2"
        X = X1 And (CDbl(X2) > 0#)
 Case "Pyro"
        X = X1 Or X2
 Case "PyroNeg"
        X = Not X1 And Not X2
 Case "Relay"
        If X1 = Not X2 Then
                              X = X1
                    Else: X = Null
                    End If
 Case "Thruster"
        X = X1 And X2 And Not X3
```

```
Case "Meng"
       X = (X1 Or X2) And (X3 Or X4)
Case "Average"
       X = (X1 + X2) / 2
Case "Average3"
       X = (X1 + X2 + X3) / 3
Case "Current"
       X = (CDbl(X1) > 0#)
Case "GasLaw"
'specific case comparing the calculated volume of a tank to the previous one
'calls Observer independently and accesses the previous values in the main table
'this case is created as a proof of extendability of the concept
'input: item name
'output: volume of the gas/liquid as double
       With rs_main
        .FindFirst ("[ITEM] ='" & item & "'")
        If Not (IsNull(!PARAM_1) Or IsNull(!PARAM_2) Or IsNull(!PARAM_3)) Then
         T = CDbl(!PARAM_1)
         P = CDbl(!PARAM_2)
         V = CDbl(!PARAM_3)
          .FindFirst ("ME")
           M = !POWERED
       X = P * CDbl(X1) * V / (CDbl(X2) * (T + 273.15))
       If Abs(X - V) > 5 And M = False Then
                        Select Case item
                          Case "ACM PT1"
                            Call Observer(item, param_id, 0, 0.001, 0.035, 0.0355, X)
                          Case "ACM PT3"
                            Call Observer(item, param_id, 0, 0.001, 0.35, 0.37, X)
                          Case "ACM PT4"
                            Call Observer(item, param_id, 0, 0.001, 0.58, 0.6, X)
                        End Select
                      End If
              Else:
                 Select Case item
                 Case "ACM PT1"
                             X = 0.025
                 Case "ACM PT3"
                             X = 0.48
                 Case "ACM PT4"
                             X = 0.25
                        End Select
              End If
```

```
            End With
        End Select
 VALEX = X
End Function
```

```
Private Sub Observer(item, param_id, LH, LS, HS, HH, X)
'Updates the main table with countdowns and keeps its own table of observed values
'input: item and param_id as Strings defining uniquely the record, LH, LS, HS and HH as
double, defining the soft and hard, lower and upper limits and the value X as Double
'output: record in the observed table, including the record of averages, simple
extrapolation result is updated into the main table
 Set rs_observed = db.OpenRecordset("observed")
 Dim param As String
 Dim T As Integer
 Dim H As Double
 Dim countdown As Double
 param = param_id
'creating a record or adding to a record
 Call Push(X, item, param_id)
  If X < LS Then H = LH - X
  If X > HS Then H = HH - X
  T = 1 'step size unimplemented

'primitive extrapolation
With rs_observed
  .FindFirst ("[NO] =" & Seeker(item, param_id))
  avgtopix = !avgtopix
  If .Fields(avgtopix - 1) <> Null Then
            countdown = H * T / (.Fields(avgtopix) - .Fields(avgtopix - 1)) - 1
                                Else:
            countdown = 99999999
                                End If
  If countdown < 0 Then countdown = 100000000
  .Edit
  !Count = countdown
  .Update
End With
'in case one item has more countdowns, the lowest one is significant
  If (Twins(item, param_id) > 0) Then
                countdown = 100000000
                With rs_observed
                .FindFirst ("[ITEM] ='" & item & "'")
                Do
                If countdown > CDbl(!Count) Then
```

```
                                    countdown = CDbl(!Count)
                                    param = !param_id
                                    End If
                .FindNext ("[ITEM] ='" & item & "'")
                Loop Until (.EOF)
                End With
              End If
' countdown is updated to Countsteps in main
  With rs_main
  .FindFirst ("[ITEM] ='" & item & "'")
  .Edit
  .Fields(6) = countdown
  .Fields(16) = param
  .Update
  End With
End Sub
```

```
Private Sub Checker()
'checker is independent of provided information, it processes the current state of the
main table
'input: main table
'output: 4 values of different checks as boolean
Set db = CurrentDb
Set rs_checked = db.OpenRecordset("qry_checked")
With rs_checked
.MoveFirst
Do
 'checker: calculates values from its lookup table and only reports if they differ from
an expected one
 '          distributed into 4  checks
 '         reporting: ruler
 item = !item
If (FAILURE_CONSEQUENCE_CHECK(item) = True) Or (FAILURE_CONDITION_CHECK(item) = True) Or
(FUNCTIONALITY_CONSEQUENCE_CHECK(item) = False) And (FUNCTIONALITY_CONDITION_CHECK(item)
= True) Then Call Ruler(item)
 If (NoInfo = True) Then Call Shower(0, item, whatswrong, "Try to obtain more data on
this part.")
 .MoveNext
Loop Until .EOF
End With
End Sub
```

```
Public Function FAILURE_CONDITION_CHECK(item) As Boolean
'reads the values of failure conditions marked by checked table inside the main table
and reasons from them
'input: item as string, checker table, main table
```

```
'output: boolean value saying whether conditions for a failure are fulfilled
Dim n As Boolean
Dim Y1 As Boolean
Dim Y2 As Boolean
Dim Y3 As Boolean
Dim Y4 As Double
Dim Y5 As Boolean
n = True
If (item = "") Then
                    n = False
Else:
With rs_checked
 .FindFirst ("[ITEM] ='" & item & "'")
 Y1 = DLookup(!FAILURE_CONDITION_1, "qry_main", "[item] ='" & item & "'")
 If Not IsNull(!FAILURE_CONDITION_2) Then Y2 = DLookup(!FAILURE_CONDITION_2, "qry_main",
"[item] ='" & item & "'")
 If Not IsNull(!FAILURE_CONDITION_3) Then Y3 = DLookup(!FAILURE_CONDITION_3, "qry_main",
"[item] ='" & item & "'")
 If Not IsNull(!FAILURE_CONDITION_4) Then Y4 = DLookup(!FAILURE_CONDITION_4, "qry_main",
"[item] ='" & item & "'")
 If Not IsNull(!FAILURE_CONDITION_5) Then Y5 = DLookup(!FAILURE_CONDITION_5, "qry_main",
"[item] ='" & item & "'")
End With
If Not IsNull(Y4) Then
 If (Y1 = False And Y2 = True Or Y1 = False And Y3 = True Or Y3 = False And Y2 = True Or
CDbl(Y4) <> 100000000 Or Y5 = True) Then
                                                                                n = True
                                                                                Else:
                                                                                n =
False
                                                                                End If
     Else:
          n = True
          whatswrong = "My table is corrupted on this place, please, use the reset
button."
    End If
End If
FAILURE_CONDITION_CHECK = n
End Function
```

```
Public Function FAILURE_CONSEQUENCE_CHECK(item) As Boolean
'reads the values of failure consequences marked by checked table inside the main table
and reasons from them
'input: item as string, checker table, main table
'output: boolean value saying whether consequences of a failure are fulfilled
Dim n1 As Boolean
```

```
Dim n2 As Boolean
Dim n3 As Boolean
Dim Y1 As Variant
Dim Y2 As Variant
Dim Y3 As Variant
Dim Y4 As Variant
Dim Query1 As String
Dim Query2 As String
n1 = True
n2 = False
n3 = True
If (item = "") Then
                n1 = False
                n2 = False
                n3 = False
           Else:
    With rs_checked
     .FindFirst ("[ITEM] ='" & item & "'")
     Y1 = DLookup(!FAILURE_CONSEQ_PARAM_ID1, "main", "[item] ='" & item & "'")
     If Not IsNull(!FAILURE_CONSEQ_ITEM2) Then Y2 = DLookup(!FAILURE_CONSEQ_PARAM_ID2,
"qry_main", "[item] ='" & !FAILURE_CONSEQ_ITEM2 & "'")
     If Not IsNull(!FAILURE_CONSEQ_ITEM3) Then Y3 = DLookup(!FAILURE_CONSEQ_PARAM_ID3,
"qry_main", "[item] ='" & !FAILURE_CONSEQ_ITEM3 & "'")
     If Not IsNull(!FAILURE_CONSEQ_ITEM4) Then Y4 = DLookup(!FAILURE_CONSEQ_PARAM_ID4,
"qry_main", "[item] ='" & !FAILURE_CONSEQ_ITEM4 & "'")
     Query1 = !FAILURE_CONSEQ_QUERY1
     If Not IsNull(!FAILURE_CONSEQ_QUERY2) Then Query2 = !FAILURE_CONSEQ_QUERY2
    Select Case Query1
    Case "Countdown"
     If Not IsNull(Y1) Then
      If CDbl(Y1) <> 100000000 Then
                       n1 = True
                       WORST = DLookup("[WORST]", "qry_main", "[item] ='" &
!FAILURE_CONSEQ_ITEM2 & "'")
                       whatswrong = "Countdown on " & WORST & " property of the item."
                      Else:
                       n1 = False
                      End If
     Else:
          n1 = False
        NoInfo = True
          whatswrong = "My table is corrupted on this place, please, use the reset
button."
    End If
```

```
    End Select
    Select Case Query2
    Case "Extreme"
       If Not IsNull(Y2) Then
        If n2 = False Then n2 = False
        Else:
        n2 = False
        NoInfo = True
        whatswrong = "No data to be compared to the limits."
        End If
    Case "Switch"
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
      If (Y2 = False And Y3 = True Or Y2 = True And Y3 = False) Then
                                                          n3 = True
                                                          If Y2 = False Then
                                                                  whatswrong =
"Switch is off, but there is a stray current."
                                                                  Else:
                                                                  whatswrong =
"Switch is on, but there is no current."
                                                                  End If
                                                          Else:
                                                          n3 = False
                                                          End If
      Else:
      n2 = False
        NoInfo = True
      whatswrong = "Not enough data to verify the switch behaviour."
      End If
    End Select
    End With
        End If
FAILURE_CONSEQUENCE_CHECK = n1 Or n2 Or n3
End Function
```

```
Public Function FUNCTIONALITY_CONDITION_CHECK(item) As Boolean
'reads the values of functionality conditions marked by checked table inside the main
table and reasons from them
'input: item as string, checker table, main table
'output: boolean value saying whether conditions for a functionality are fulfilled
Dim n As Boolean
Dim Y1 As Boolean
Dim Y2 As Boolean
Dim Y3 As Boolean
```

```
n = False
If (item = "") Then
                    n = True
                Else:
With rs_checked
 .FindFirst ("[ITEM] ='" & item & "'")
 Y1 = DLookup(!FUNC_CONDITION_1, "main", item = item)
 If Not IsNull(!FUNC_CONDITION_2) Then Y2 = DLookup(!FUNC_CONDITION_2, "qry_main",
"[item] ='" & item & "'")
 If Not IsNull(!FUNC_CONDITION_3) Then Y3 = DLookup(!FUNC_CONDITION_3, "qry_main",
"[item] ='" & item & "'")
End With
If Y1 = True And Y2 = True And Y3 = True Then
                                    n = True
                                    Else:
                                    n = False
                                    End If
End If
FUNCTIONALITY_CONDITION_CHECK = n
End Function
```

```
Public Function FUNCTIONALITY_CONSEQUENCE_CHECK(item) As Boolean
'reads the values of functionality consequences marked by checked table inside the main
table and reasons from them
'input: item as string, checker table, main table
'output: boolean value saying whether consequences of a functionality are fulfilled
Dim n1 As Boolean
Dim n2 As Boolean
Dim n3 As Boolean
Dim n4 As Boolean
Dim Firing As Boolean
Dim Y1 As Variant
Dim Y2 As Variant
Dim Y3 As Variant
Dim Y4 As Variant
Dim Query1 As String
Dim Query2 As String
Dim Query3 As String
Dim Query4 As String
Dim MINI As Double
Dim MAXI As Double
MINI = 0
MAXI = 30
n1 = False
```

```
n2 = False
n3 = False
n4 = False
Firing = False
If IsNull(item) Then
                n1 = True
                n2 = True
                n3 = True
                n4 = True
            Else:
    With rs_checked
     .FindFirst ("[ITEM] ='" & item & "'")
     Y1 = DLookup(!FUNC_CONSEQ_PARAM_ID1, "qry_main", "[item] ='" & !FUNC_CONSEQ_ITEM1 &
"'")
     If Not IsNull(!FUNC_CONSEQ_ITEM2) Then Y2 = DLookup(!FUNC_CONSEQ_PARAM_ID2,
"qry_main", "[item] ='" & !FUNC_CONSEQ_ITEM2 & "'")
     If Not IsNull(!FUNC_CONSEQ_ITEM3) Then Y3 = DLookup(!FUNC_CONSEQ_PARAM_ID3,
"qry_main", "[item] ='" & !FUNC_CONSEQ_ITEM3 & "'")
     If Not IsNull(!FUNC_CONSEQ_ITEM4) Then Y4 = DLookup(!FUNC_CONSEQ_PARAM_ID4,
"qry_main", "[item] ='" & !FUNC_CONSEQ_ITEM4 & "'")
     Query1 = !FUNC_CONSEQ_QUERY1
     If Not IsNull(!FUNC_CONSEQ_QUERY2) Then Query2 = !FUNC_CONSEQ_QUERY2
     If Not IsNull(!FUNC_CONSEQ_QUERY3) Then Query3 = !FUNC_CONSEQ_QUERY3
     If Not IsNull(!FUNC_CONSEQ_QUERY4) Then Query4 = !FUNC_CONSEQ_QUERY4
     Select Case Query1
     Case "Default"
     If Not IsNull(Y1) Then
           n1 = Y1
           If n1 = False Then whatswrong = !FUNC_CONSEQ_PARAM_ID1
      Else: n1 = True
            NoInfo = True
           whatswrong = "Not enough data to perform a functionality check on this item."
      End If
     Case "Temp"
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
                n1 = (Abs(Y1 - Y2) < 10 And Abs(Y2 - Y3) < 10)
                If n1 = False Then
                        If !FUNC_CONSEQ_PARAM_ID1 = "PARAM_1" Then whatswrong =
"Temperatures in the near area are uncorrelated."
                        If !FUNC_CONSEQ_PARAM_ID1 = "PARAM_2" Then whatswrong =
"Pressures in the near area are uncorrelated."
                End If
       Else:
                n1 = True
```

```
                NoInfo = True
                whatswrong = "This sensor or one in the near surrounding is not
delivering any data."
       End If


     Case "Pres"
     If Not IsNull(Y1) Then
       n1 = (CDbl(Y1) > 0)
       If n1 = False Then
                        If Left(!item, 2) = "AC" Then whatswrong = "There seems to be no
pressure in the lower part of the system."
                        If Left(!item, 2) = "TH" Then whatswrong = "The thruster hasn't
thrusted yet or there is no information of it."
                     End If
        Else:
                n1 = True
                NoInfo = True
                whatswrong = "This sensor or one in the near surrounding is not
delivering any data."
       End If
     Case "Latch"
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
       n1 = (Abs(Y1 - Y2) < 2 And Abs(Y1 - Y3) < 2)
       If n1 = False Then whatswrong = "The latch valves need to be opened, as the
pressure difference above and under them is higher than limits."
       Else:
                n1 = True
                NoInfo = True
                whatswrong = "One of the pressure sensors in the near surrounding is not
delivering any data."
       End If
     Case "GasLaw"
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
       n1 = (MINI < Y3 * Y2 / (Y1 + 273) < MAXI)
       If n1 = False Then whatswrong = "Pressure and temperature are not correlated with
the expected volume."
       Else:
                n1 = True
                NoInfo = True
                whatswrong = "Not enough information to calculate volume, temperature or
pressure sensor didn't deliver the data."
       End If
     Case "GasLaw2"
     If Not (IsNull(Y1) Or IsNull(Y2)) Then
       n1 = (Y2 And (CDbl(Y1) > 1) Or Not Y2)
```

```
       If n1 = False Then whatswrong = "The local heater is on, but the temperature is
very low."
       Else:
                n1 = True
                NoInfo = True
                whatswrong = "There is no data on the status of the heater or the
temperature."
       End If
    End Select
    Select Case Query2
     Case "Default"
     If Not IsNull(Y2) Then
           n2 = Y2
           If n2 = False Then whatswrong = !FUNC_CONSEQ_PARAM_ID2
      Else: n2 = True
                NoInfo = True
           whatswrong = "Not enough data to perform a functionality check on this item."
      End If
     Case "Temp"
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
                n2 = (Abs(Y1 - Y2) < 10 And Abs(Y2 - Y3) < 10)
                If n2 = False Then
                        If !FUNC_CONSEQ_PARAM_ID2 = "PARAM_1" Then whatswrong =
"Temperatures in the near area are uncorrelated."
                        If !FUNC_CONSEQ_PARAM_ID2 = "PARAM_2" Then whatswrong =
"Pressures in the near area are uncorrelated."
                End If
       Else:
                n2 = True
                NoInfo = True
                whatswrong = "This sensor or one in the near surrounding is not
delivering any data."
       End If
     Case "Latch"
      If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
       n2 = (Abs(Y1 - Y2) < 2 And Abs(Y1 - Y3) < 2)
       If n2 = False Then whatswrong = "The latch valves need to be opened."
       Else:
                n2 = True
                NoInfo = True
                whatswrong = "One of the pressure sensors in the near surrounding is not
delivering any data."
       End If
     Case "GasLaw"
```

```
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
       n2 = (MINI < Y3 * Y2 / (Y1 + 273) < MAXI)
       If n2 = False Then whatswrong = "Pressure and temperature are not correlated with
the expected volume."
       Else:
                n2 = True
                NoInfo = True
                whatswrong = "Not enough information to calculate volume, temperature or
pressure sensor didn't deliver the data."
        End If
     Case "GasLaw2"
     If Not (IsNull(Y1) Or IsNull(Y2)) Then
       n2 = (Y2 And (CDbl(Y1) > 1) Or Not Y2 And (CDbl(Y1) < 50))
       If n2 = False Then
                        If Y2 = True Then whatswrong = "The local heater is on, but the
temperature is very low."
                        If Y2 = False Then whatswrong = "The local heater is off, but
the temperature is rather high."
                        End If
       Else:
                n2 = True
                NoInfo = True
                whatswrong = "There is no data on the status of the heater or the
temperature."
       End If
     Case "OnFire"
      If Not IsNull(Y2) Then
       n2 = (CDbl(Y2) > 100)
       If n2 = False Then whatswrong = "The Main Engine seems not to be firing."
       Else:
                n2 = True
                NoInfo = True
                whatswrong = "Not enough information to evaluate whether ME is firing."
       End If
    End Select
    Select Case Query3
     Case "Default"
     If Not IsNull(Y3) Then
           n3 = Y3
           If n3 = False Then whatswrong = !FUNC_CONSEQ_PARAM_ID3
      Else:
            n3 = True
                NoInfo = True
```

```
            whatswrong = "Not enough data to perform a functionality check on this
item."
      End If
     Case "Temp"
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
                n3 = (Abs(Y1 - Y2) < 10 And Abs(Y2 - Y3) < 10)
                If n3 = False Then
                        If !FUNC_CONSEQ_PARAM_ID3 = "PARAM_1" Then whatswrong =
"Temperatures in the near area are uncorrelated."
                        If !FUNC_CONSEQ_PARAM_ID3 = "PARAM_2" Then whatswrong =
"Pressures in the near area are uncorrelated."
                End If
       Else:
                n1 = True
                NoInfo = True
                whatswrong = "This sensor or one in the near surrounding is not
delivering any data."
       End If
     Case "Latch"
      If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
       n3 = (Abs(Y1 - Y2) < 2 And Abs(Y1 - Y3) < 2)
       If n3 = False Then whatswrong = "The latch valves need to be opened."
       Else:
                n3 = True
                NoInfo = True
                whatswrong = "One of the pressure sensors in the near surrounding is not
delivering any data."
       End If
     Case "GasLaw"
     If Not (IsNull(Y1) Or IsNull(Y2) Or IsNull(Y3)) Then
       n3 = (MINI < Y3 * Y2 / (Y1 + 273) < MAXI)
       If n3 = False Then whatswrong = "Pressure and temperature are not correlated with
the expected volume."
       Else:
                n3 = True
                NoInfo = True
                whatswrong = "Not enough information to calculate volume, temperature or
pressure sensor didn't deliver the data."
       End If
    End Select
    Select Case Query4
     Case "Default"
     If Not IsNull(Y4) Then
           n4 = Y4
           If n4 = False Then whatswrong = !FUNC_CONSEQ_PARAM_ID4
```

```
      Else:
            n4 = True
            NoInfo = True
           whatswrong = "Not enough data to perform a functionality check on this item."
      End If
     Case "OnFire"
      If Not IsNull(Y4) Then
       n4 = (CDbl(Y4) > 100)
       n4 = (CDbl(Y4) > 100)
       Firing = (CDbl(Y4) > 100)
       If n4 = False Then whatswrong = "The thruster seems not to be firing."
       Else:
                n4 = True
                NoInfo = True
                whatswrong = "Not enough information to evaluate whether this thruster
is firing."
       End If
    End Select
    End With
        End If
FUNCTIONALITY_CONSEQUENCE_CHECK = (n1 And n2 And n3 And n4) Or Firing
End Function
```

```
Private Sub Reloader()
Dim itemid As String
 Set db = CurrentDb
 Set rs_relevant = db.OpenRecordset("qry_relevant")
 Set rs_main = db.OpenRecordset("qry_main")
 Set rs_observed = db.OpenRecordset("qry_observed")
 Set rs_records = db.OpenRecordset("records")
'puts main table back in the default state (empty, altruistic values set to True) and
resets the packet counters in relevant table
'input: button pressed
'output: changes in the main table and relevant table
With rs_main
 .MoveFirst
 Do
   For i = 1 To 4
   .Edit
   .Fields(i) = False
   .Update
  Next
  .Edit
  .Fields(5) = 100000000
```

```
.Fields(6) = 100000000
.Update
For i = 7 To 16
 .Edit
 .Fields(i) = Null
 .Update
Next
itemid = Left(!item, 2)
Select Case itemid
 Case "TE"
  .Edit
  .Fields(2) = True
  .Fields(3) = True
  .Update
 Case "TH"
  .Edit
  .Fields(1) = True
  .Fields(2) = True
  .Update
 Case "ME"
  .Edit
  .Fields(2) = True
  .Update
 Case "LC"
  .Edit
  .Fields(1) = True
  .Update
 Case "RT"
  .Edit
  .Fields(1) = True
  .Update
 Case "FC"
  .Edit
  .Fields(2) = True
  .Update
 Case "AC"
  .Edit
  .Fields(2) = True
  .Update
 Case "NT"
  .Edit
  .Fields(1) = True
  .Fields(2) = True
```

```
    .Fields(3) = True
    .Update
   Case "MM"
    .Edit
    .Fields(1) = True
    .Fields(2) = True
    .Fields(3) = True
    .Update
   Case "HE"
    .Edit
    .Fields(1) = True
    .Fields(2) = True
    .Fields(3) = True
    .Update
   Case "PV"
    .Edit
    .Fields(1) = True
    .Update
'default state
  End Select
  .MoveNext
 Loop Until (.EOF)
End With
With rs_relevant
 .MoveFirst
 Do
  .Edit
  !LAST_PACKETNR = 0
  .Update
  .MoveNext
 Loop Until (.EOF)
End With
With rs_records
 If .RecordCount > 0 Then
 .MoveFirst
  Do
  .Delete
  .MoveNext
  Loop Until (.EOF)
 End If
 End With
 With rs_observed
 If .RecordCount > 0 Then
```

```
 .MoveFirst
  Do
  .Delete
  .MoveNext
  Loop Until (.EOF)
  End If
 End With
End Sub
```

```
Private Sub Ruler(item As String)
Dim failure As Integer
Dim recovery As String
 'ruler: uses the if - then to estimate situation
 '       uses Recovery lookup table to output
 '       calls shower
 '0)  Check conditions necessary for failure occurrence.
'a.  YES -> it is an error
'b.  NO -> it is not an error (it is a phantom error)
'1)  Check settings necessary for functionality state.
'a.  YES -> it is not a process error, it is a functionality error
'b.  NO -> it is a process error
'2)  Check HW preceding and subsequent failures in hierarchy for identification.
'a.  Preceding fulfilled  -> move up in hierarchy, likely NOT an error here
'b.  Subsequent fulfilled -> HW error
'c.  Subsequent not fulfilled -> simply weird functionality, repeating the command
recommended
'input: the output of the Checker
'output: number of a failure type as integer
Call Loader(item)
 If (FAILURE_CONSEQUENCE_CHECK(item) = True) Or (FUNCTIONALITY_CONSEQUENCE_CHECK(item) =
False) And (FUNCTIONALITY_CONDITION_CHECK(item) = True) Then
    If FAILURE_CONDITION_CHECK(item) = True Then
                    'it is an error
                    failure = 1
                    If FUNCTIONALITY_CONDITION_CHECK(item) = True Then
                              'functionality error
                              failure = 2
                              If (FAILURE_CONSEQUENCE_CHECK(SUBSEQUENT1) = True) And
(FAILURE_CONDITION_CHECK(SUBSEQUENT1) = True) Or (FAILURE_CONSEQUENCE_CHECK(SUBSEQUENT2)
= True) And (FAILURE_CONDITION_CHECK(SUBSEQUENT2) = True) Or
(FAILURE_CONSEQUENCE_CHECK(SUBSEQUENT3) = True) And
(FAILURE_CONDITION_CHECK(SUBSEQUENT3) = True) Or (FAILURE_CONSEQUENCE_CHECK(SUBSEQUENT4)
= True) And (FAILURE_CONDITION_CHECK(SUBSEQUENT4) = True) Then
                                        'HW failure
                                        failure = 4
                                        With rs_main
```

```
                                        .FindFirst ("[item] = '" & item & "'")
                                        .Edit
                                        !FAILED = True
                                        .Update
                                        End With
                                        Else:
                                        'not a failure
                                        failure = 0
                                        End If
                                If (FAILURE_CONSEQUENCE_CHECK(PRECEDING1) = True) And
(FAILURE_CONDITION_CHECK(PRECEDING1) = True) Or (FAILURE_CONSEQUENCE_CHECK(PRECEDING2) =
True) And (FAILURE_CONDITION_CHECK(PRECEDING2) = True) Or
(FAILURE_CONSEQUENCE_CHECK(PRECEDING3) = True) And (FAILURE_CONDITION_CHECK(PRECEDING3)
= True) Or (FAILURE_CONSEQUENCE_CHECK(PRECEDING4) = True) And
(FAILURE_CONDITION_CHECK(PRECEDING4) = True) Then failure = 0
                                If (FUNCTIONALITY_CONDITION_CHECK(item) = True) And
(FUNCTIONALITY_CONSEQUENCE_CHECK(item) = True) Then failure = 0
                                Else:
                                'process error
                                failure = 3
                                End If
Else:
'not an error, but will be mentioned as "seemingly faulty"
failure = 0
End If
 End If
If failure = 0 Then
    recovery = "Examine the suspected part."
                Else:
    recovery = DLookup("[recovery]", "qry_recovery", "[NO] =" & Sucher(item, failure))
    End If
param = param_id
Call Shower(failure, item, whatswrong, recovery)
End Sub
```

```
Public Function Sucher(item, failure)
'returns the number of record where item = item and failure = failure in the recovery
table, same as Seeker
'input: global variables item as String and failure number as integer
'output: record number as integer
With rs_recovered
Dim n As Long
n = 0
.MoveFirst
.FindFirst ("[ITEM] ='" & item & "'")
Do
```

```
  If (!failure = failure) Then
                                    n = !NO
                                    Exit Do
                               Else:
                                    .FindNext ("[ITEM] ='" & item & "'")
                                    End If
Loop Until (.EOF)
End With
Sucher = n
End Function
```

```
Public Sub Loader(item)
 'reading the relevant parameter names (String) from the relevant table generated lookup
table
 'looking up the values of the parameters in VEX_TM
 'input: table readings
 'output: values X1 to X6 as Variant, parameters of a given item
  PRECEDING1 = ""
 SUBSEQUENT1 = ""
  PRECEDING2 = ""
 SUBSEQUENT2 = ""
  PRECEDING3 = ""
 SUBSEQUENT3 = ""
  PRECEDING4 = ""
 SUBSEQUENT4 = ""
With rs_checked
.FindFirst ("[ITEM] ='" & item & "'")
If Not IsNull(!PRECEDING1) Then PRECEDING1 = rs_checked!PRECEDING1
If Not IsNull(!SUBSEQUENT1) Then SUBSEQUENT1 = rs_checked!SUBSEQUENT1
If Not IsNull(!PRECEDING2) Then PRECEDING2 = rs_checked!PRECEDING2
If Not IsNull(!SUBSEQUENT2) Then SUBSEQUENT2 = rs_checked!SUBSEQUENT2
If Not IsNull(!PRECEDING3) Then PRECEDING3 = rs_checked!PRECEDING3
If Not IsNull(!SUBSEQUENT3) Then SUBSEQUENT3 = rs_checked!SUBSEQUENT3
If Not IsNull(!PRECEDING4) Then PRECEDING4 = rs_checked!PRECEDING4
If Not IsNull(!SUBSEQUENT4) Then SUBSEQUENT4 = rs_checked!SUBSEQUENT4
End With
End Sub
```

```
Private Sub Shower(failure As Integer, item As String, whatswrong As String, recovery As
String)
Set rs_records = db.OpenRecordset("records")
 'ruler calls a query that reads out the failure type recovery information for the given
item/case from the recovery table and passes it to the shower
 'shower: displays output of the ruler as a warning box and a list box (and puts it into
the records table) along with the query result for the relevant recovery text
```

```
 'input: failure number as integer, item name and recovery text
 'output: records table and screen
 Call Looser(failure)
 With rs_main
 .FindFirst ("[ITEM] ='" & item & "'")
 If failure > 1 Then
  If (!countsteps = 100000000) Then
   MsgBox Looser(failure) & " failure detected on part " & item & ". " & whatswrong & "
Recommended recovery procedure: " & recovery, vbOKOnly
   Else:
   countsteps = !countsteps
   MsgBox Looser(failure) & " failure detected on part " & item & ". " & whatswrong & "
Fatal failure in " & countsteps & " steps. Recommended recovery procedure: " & recovery,
vbOKOnly
  End If
 End If
End With
With rs_records
 .AddNew
 !TIME = TIME
 !item = item
 !WRONG = whatswrong
 !failure = Looser(failure)
 !recovery = recovery
 .Update
 End With
End Sub
```

```
Private Function Looser(failure) As String
'translates failure number into words
'input: failure number as integer
'output: string
Dim n As String
Select Case failure
Case 0
    n = "Potential"
Case 1
    n = "Certain"
Case 2
    n = "Functionality"
Case 3
    n = "Process"
Case 4
    n = "Hardware"
```

```
End Select
Looser = n
End Function
```

```
Public Function Seeker(item, param_id) As Long
'returns the number of record where item = item and param_id = param_id
'input: global strings item and param_id, observed table contents
'output: number of record as integer
 Dim n As Long
 Dim counterseek As Long
 Set rs_observed = db.OpenRecordset("observed")
n = 0
With rs_observed
If counterseek = 0 Then
                    n = 0
                    Else:
            .MoveFirst
            .FindFirst ("[ITEM] ='" & item & "'")
            Do
             If (!param_id = param_id) Then
                                n = !NO
                                Exit Do
                           Else:
                                .FindNext ("[ITEM] ='" & item & "'")
                                End If
            Loop Until (.EOF)
        End If
End With
counterseek = counterseek + 1
Seeker = n
End Function
```

```
Public Function Twins(item, param_id) As Integer
'checks if the item does have more than one countdowns related
'input: global strings item and param_id, observed table contents
'output: number of other records for item as integer
Dim n As Integer
n = 0
With rs_observed
.MoveFirst
.FindFirst ("[ITEM] ='" & item & "'")
Do
  If (!param_id <> param_id) Then n = n + 1
  .FindNext ("[ITEM] ='" & item & "'")
```

```
Loop Until (.EOF)
End With
Twins = n
End Function
```

```
Public Sub NewQueue(item, param_id)
'creates a new empty record in the DB
'input: item, param_id
'output: a record in observed table
With rs_observed
.AddNew
!item = item
!param_id = param_id
!topix = 3
For i = 3 To 12
.Fields(i) = Null
Next
For i = 15 To 24
.Fields(i) = Null
Next
!bumix = 3
!avgtopix = 15
!avgbumix = 15
.Update
End With
End Sub
```

```
Public Sub Push(VALEX, item, param_id)
'puts the value and the new average on its place in a record
'input: value as double, global strings item and param_id
'output: update of a record in observed table
With rs_observed

If (Seeker(item, param_id) = 0) Then
                Call NewQueue(item, param_id)
                .MoveLast
                n = !NO
                Else:
                .FindFirst ("[NO] =" & Seeker(item, param_id))
                End If
topix = !topix
bumix = !bumix
If IsFull(item, param_id) Then Call Pop(item, param_id)
.Edit
```

```
.Fields(topix) = VALEX
.Update
If topix + 1 > 12 Then
                        .Edit
                        !topix = 3
                        .Update
                    Else:
                        .Edit
                        !topix = topix + 1
                        .Update
                        End If
If Seeker(item, param_id) <> 0 Then .FindFirst ("[NO] =" & Seeker(item, param_id))
    avgtopix = !avgtopix
     .Fields(avgtopix) = Average(item, param_id)
      If avgtopix + 1 > 24 Then
                              .Edit
                              !avgtopix = 15
                              .Update
                        Else:
                              .Edit
                              !avgtopix = avgtopix + 1
                              .Update
                        End If
End With
End Sub
```

```
Public Function IsEmpty(item, param_id) As Boolean
'returns True if no number is entered to the record
'input: global strings item and param_id, observed table contents
'output: boolean info on the contents of the table
Dim n As Boolean
With rs_observed
If Seeker(item, param_id) = 0 Then
                                n = True
                               Else:
                                .FindFirst ("[NO] =" & Seeker(item, param_id))
                                n = (!bumix = !topix) And (!Fields(topix + 1) = Null)
End If
End With
IsEmpty = n
End Function
```

```
Public Function IsFull(item, param_id) As Boolean
'returns True if all 10 values of given record are filled with numbers
```

```
'input: global strings item and param_id, observed table contents
'output: boolean info on the contents of the table
Dim n As Boolean
With rs_observed
If Seeker(item, param_id) = 0 Then
                                n = False
                               Else:
                                .FindFirst ("[NO] =" & Seeker(item, param_id))
                                topix = !topix
                                If .Fields(topix + 1) <> Null Or topix = 12 Then n = True
End If
IsFull = n
End With
End Function
```

```
Public Sub Pop(item, param_id)
'removes the beginning of the queue
'input: global strings item and param_id, observed table contents
'output: deleting part of contents of the observed table
If Seeker(item, param_id) <> 0 Then
With rs_observed
.Edit
.FindFirst ("[NO] =" & Seeker(item, param_id))
 bumix = !bumix
 .Fields(bumix) = Null
 If bumix + 1 > 12 Then
                           !bumix = 3
                     Else: !bumix = bumix + 1
                     End If
 .Fields(avgbumix) = Null
 If avgbumix + 1 > 24 Then
                           !avxbumix = 15
                     Else: !avgbumix = bumix + 1
                     End If
.Update
End With
End If
End Sub
```

```
Public Sub EmptyQ(item, param_id)
'empties all the value numbers in the record
'input: global strings item and param_id, observed table contents
'output: deleting part of contents of the observed table
Do
```

```
Call Pop(item, param_id)
Loop Until IsEmpty(item, param_id)
End Sub
```

```
Public Function Sum(item, param_id) As Double
'returns the sum of existing values for given record for average calculation
'input: global strings item and param_id, observed table contents
'output: sum of the record values as double
Dim n As Double
If Seeker(item, param_id) <> 0 Then
With rs_observed
.FindFirst ("[NO] =" & Seeker(item, param_id))
n = 0
topix = !topix
If IsFull(item, param_id) Then
            For i = 3 To 12
            n = n + !Fields(i)
            Next
          Else:
            For i = 1 To topix
            n = n + !Fields(i)
            Next
    End If
End With
                Else:
                n = Null
End If
Sum = n
End Function
```

```
Public Function Average(item, param_id) As Double
'returns the average of the record values
'input: global strings item and param_id, observed table contents
'output: average of the record values as double
Dim n As Double
If Seeker(item, param_id) <> 0 Then
With rs_observed
.Edit
.FindFirst ("[NO] =" & Seeker(item, param_id))
If IsFull(item, param_id) Then
                    n = Sum(item, param_id) / 10
                          Else:
                    n = Sum(item, param_id) / !topix
    End If
```

```
End With
                    Else:
                    n = VALEX(item, param_id)
                    End If
Average = n
End Function
```

```
Private Sub ButtonReset_Click()
Call Reloader
End Sub
```

```
Private Sub ButtonStart_Click()
  Form_Timer
End Sub
```