

THE MODEL BASED MISSION CONTROL SYSTEM

K. Bradbury* & P. Holding** & J. Wheadon***

* *Logica UK Ltd. Stephenson House, 75 Hampstead Road, London NW1 2PL*

Fax: +44 171 468 7006, E-mail: bradburyk@logica.com

** *Science System (Space) Ltd. 23 Clothier Road, Brislington, Bristol, BS4 5PS*

Fax: +44 117 971 1125, E-mail: holding_pr@scisys.co.uk

*** *ESA/ESOC, Robert Bosch Str. 5, D-64293 Darmstadt, Germany,*

Fax: +49 6151 903010, E-mail: jwheadon@esoc.esa.de

ABSTRACT. The Model Based Mission Control System Study (MBMCS) was sponsored by the European Space Agency to facilitate the application of comprehensive domain modelling within future mission control systems. The study has built on previous work, [AMFESYS.93], [SCOS-11.94], and has produced an architecture for a Mission Model to be embedded in a model-based mission control system (MCS). (This architecture will be applied in a prototype of an advanced MCS being developed in another ESA study [ATOS-4.95]).

This paper outlines how a Mission Model would be used in an advanced MCS, and describes the architecture of the Mission Model developed by the MBMCS. The paper then describes the MBMCS Prototype Model Editor, also developed in this study. The Model Editor is a graphics-oriented tool which can be used by spacecraft operations engineers to create a Mission Model from a library of generic Model Elements, expressing engineering information in a natural way.

1. MISSION MODELS

The trend in modern spacecraft design is to increase the complexity of functioning of the on-board systems, largely by the use of embedded programmable processors controlling various automatic sequences/cycles and modes of operation, for example in the management of the spacecraft platform and payload housekeeping, and for performance of experiments or other payload services.

The European Space Operations Centre (ESOC) have in the past developed re-usable MCS software to support a number of different missions, where the MCS was configurable largely through the setting up of tables of characteristics of each spacecraft, supplemented by specially-coded mission-specific software applications. These tables constituted the "knowledge" of the spacecraft used by standardised applications of the MCS, basically for telemetry monitoring, and telecommanding.

This was a very abstract way to represent a model of the spacecraft, but the scope of the knowledge was limited to declarations of individual telemetry parameters and telecommands, and some simple checks associated with these.

ESOC's Multi Satellite Support System (MSSS), and post 1991, the Spacecraft Control and Operations System (SCOS-1), have provided partially-generic platforms from which to develop a complete MCS in this way.

These systems have been successful, especially for supporting operations of spacecraft of limited complexity. For more complex missions, such as Eureka or ERS, each MCS (based on SCOS-1), has included a significant amount of mission specific software to meet the support requirements. This fact, the availability of applicable new technologies and ideas, plus strategic considerations, has led ESOC to embark on development of new customisable MCS software known as SCOS-II, which in the long term will be much more generic.

A key to the achievement of increased genericity is seen as being the concept of a “Mission Model”, part of a Mission Information Base (MIB) which should contain all information about the operation of the mission, including the current status of the mission, and its past history.

The Mission Model should encapsulate a description of the design and functional characteristics of the spacecraft and relevant ground equipment, plus a description of all the rules, constraints and procedures applying to the operation of the mission. The aim is to provide a means whereby this information can be set up easily by mission operations engineers, and to reduce as far as possible the amount of mission-specific program code which has to be developed in standard programming language (e.g. C++) for embedding in the MCS.

The SCOS-II architectural concept foresees the use of such a Mission Model, and the MBMCS study (subject of this paper) has developed the concept further to a state where it can be used in an advanced MCS. The ATOS-4 study is applying the MBMCS results in, in a number of “advanced applications” in a prototype MCS [ATOS-4.95].

For this approach, and these new generic infrastructure developments to bring greatest economical benefits, there is a need to apply standardised ways of operating different missions as far as possible. This question is related closely to the definition of mission operations support requirements. However, this topic was not a subject of the present study

1.1 MISSION MODEL APPLICATIONS

In this paper (and indeed in the MBMCS study), we focus on development and use of models of spacecraft. The concepts are equally applicable to models of the ground segment facilities used for support of a mission.

Important applications for a spacecraft Mission Model are:

- **Telemetry Monitoring.** A Mission Model can be used to simulate in real-time the status of the spacecraft, and thereby provide reference values for checking the downlinked telemetry corresponding to any required moment of time. As the model state (inevitably) deviates from that of the real spacecraft over time, its evolution can be constantly adjusted by judicious (small) corrections to a number of key internal variables, e.g. heating/cooling rates, efficiency parameters of the power supply subsystem, etc., so long as these adjustments lie within acceptable bounds [AMFESYS.93]. The model provides a self-consistent time-evolving view to the operator (and to other applications) of the state of the spacecraft, in a way not provided by the downlinked telemetry whose individual parameters are sampled over a period of time, an inconsistency which tends to become exacerbated with use of packetised telemetry “reporting by exception”. The modelled state is available at all times, whether the spacecraft is within coverage of a ground station or not (a typical low-flying polar orbiter like ERS is visible for about 10 minutes in each 90-minute orbit).
- **Schedule Planning and Verification.** Typically, operations schedules are constructed off-line from the MCS from mission services and spacecraft housekeeping requests co-ordinated within a mission planning system using its own local store of mission reference information. In an integrated MCS which would include a mission planning/scheduling subsystem to generate detailed schedules/timelines of activities including spacecraft commanding, the Mission Model would provide knowledge of possible state-transitions of the spacecraft, for use by the planning function to select appropriate procedures of predefined commands, or even to generate these (classical AI planning problem). The Mission Model could also provide constraints and resource information to the planning/scheduling subsystem. The Mission Model could be used

subsequently in a “simulation mode”, to validate the generated schedule by executing it in “fast timing mode” and checking for violations of operating rules or constraints.

- **Telecommand Management.** Pre- and Post- Telecommand management and checking is not a new concept in an MCS. What is new is the possibility to use a Mission Model to enhance the predictive capabilities by simulation to take into account the effect of scheduled telecommands, which may be already uplinked to an on board queue.
- **Anomaly Diagnosis.** The telemetry monitoring application may detect an anomalous discrepancy between the simulated and received telemetry streams. A Diagnosis application, using information from the Mission Model describing possible state transition, inter-dependencies between on-board units, operating characteristics, plus the history of the mission, in particular recent status of the spacecraft, and telecommand history, can form hypotheses about the cause(s) of the anomaly, and can also validate these by using the Mission Model for simulation.

All of the above “advanced model-based applications” of are being developed for ESOC in the prototype ATOS-4 MCS [ATOS-4.95] which is making use of the Mission Model architecture and the Model editor developed in the MBMCS study, reported by this paper.

2. MISSION MODEL DEFINITION

The MBMCS study consolidated numerous disparate pieces of work which had investigated modelling, and a number of concepts introduced in outline by the SCOS-II development, but not taken any further. Previous ESOC studies had produced models for demonstration purposes and there is a wealth of simulator experience at ESOC. This, and investigations into the type of modelling problems that would exist led to the MBMCS study developing a structure of Mission Model that contains static design information can be used dynamically in a simulation mode, and is created and displayed graphically, as shown in figure 1:

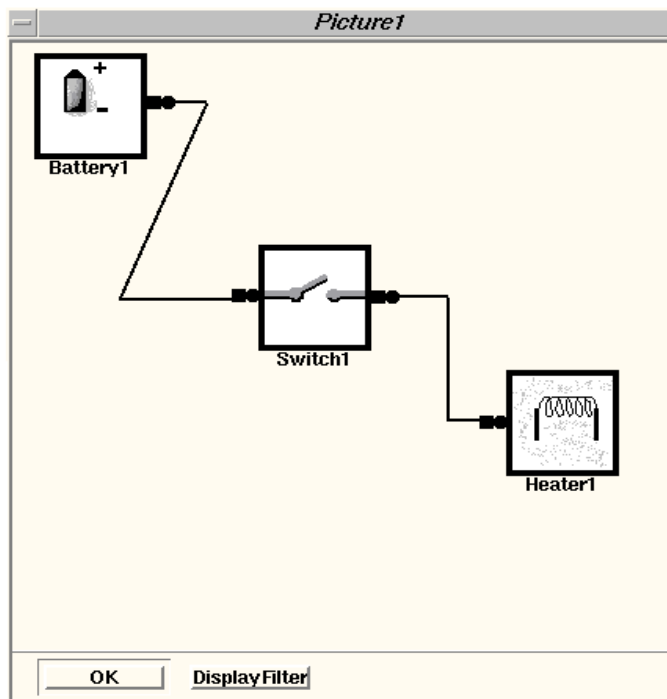


figure 1: a simple Mission Model

This diagram is created by the MBMCS Mission Model Editor, described below. The figure shows part of a simple mission model comprising a `Battery`, a `Switch` and a `Heater`, formally called *Model Elements* in MBMCS, with their interconnections, formally called *relationships*, which in this example imply a type of functional interdependency between the Model Elements. For example, should the `Switch` in the real spacecraft be turned off by a command, then the `Heater` will cool down.

The Model Elements of the model are described in natural domain terms, the creators of the Mission Model are presented with meaningful objects with which to construct a model, and not a set of function names in a library, or an API. The model is created graphically, and the same model definition is used for the graphical displays used by the MCS to show the state of each *instance* of the Mission Model used by the various applications.

2.1 MISSION MODEL COMPONENTS

An MBMCS Mission Model is constructed from object oriented classes representing Model Elements. Each Model Element is defined generically in terms of its *Attributes*, for example a `TapeRecorder` may include `TapeLength`, `TapePosition` and `RampUpTime` as attributes. Other examples of attributes include the terminals that a Model Element may have: the `TapeRecorder` may have an input `Power Terminal`, and an input and output `Data Terminal`. These terminals allow users of the Model Element to connect it to other Model Elements, such as a power source, using a `Relationship`.

A second important concept is that of *Modes* and *ModeStates*. These are used by engineers to abstract the state of the attributes, for example to represent that a `Heater` has over-heated, or that a `Switch` is `On`. Each Model Element may have a number of `Modes`, for example a `Behavioural Mode` representing physical characteristics of the Model Elements and therefore dependent upon its attributes, or a `Commanded Mode` which reflects a state change caused by a command. Each `Mode` may then contain a number of mutually exclusive `ModeStates`, for example “`TooCold`”, “`Nominal`” or “`TooHot`”.

Once a set of Model Elements have been created then they may be added to a `Palette` of Model Elements for use in Mission Models: the Model Element classes are instantiated, and therefore re-used, many times to form Mission Models.

2.2 MISSION MODEL RELATIONSHIPS

Once Model Elements have been defined then may be used to construct a Mission Model. The Model Elements are “connected” to each other using the Editor which then creates a `Mission Model Relationship` between the Model Elements. A relationship abstracts Model Elements from each other: a `Switch` does not know what it powers, but that there is a power relationship between it and a set of other Model Elements. If the state of the `Switch` changes then it is the relationship propagates the state change, not the `Switch`. Correspondingly, a `TapeRecorder` does not know how it is powered, but that it is powered. The relationship informs the `TapeRecorder` of a state change.

There are a number of identified relationships which may be used in an MBMCS Mission Model:

- **Containment:** an important relationship which allows Model Elements to exist within other Model Elements, for example a `Heater` may well be contained within a `HeaterUnit` which itself may be contained within a spacecraft model. This allows users to develop complex models piece by piece.

- Power: a power relationship may be defined between Model Elements. A power relationship “knows” about loads and drains, and the propagation of a binary status indicating if a Model Element is powered or not. It should be noted that a relationship may also be used to propagate analogue values as well as a binary status change.
- CommandFlow: allows a developer of the Mission Model to define the path that a command takes through the system. This is particularly useful to model the command processing that takes place in a mission.
- Redundancy relationships may be created so that an application can inquire of the model if a Model Element is redundant with another.
- DataFlow, Mechanical and Thermal relationships: these exist to model other types of relationship that may be desired in a Mission Model.

The concept of a relationship has been introduced so that Model Elements may be developed independently of their use, so that applications may navigate through a Mission Model by asking Model Elements about the relationships they have with others and so that state changes may be propagated through the Mission Model.

2.3 MISSION MODEL BEHAVIOUR

The MBMCS Mission Model includes behavioural information which describes how the model changes with time. The model includes a scheduler which is used to update Model Elements at defined epochs: the developer of the Model Element specifies when the Model Element is to be updated by registering the Model Element with the scheduler for immediate update, or for updates at the current epoch plus a specified period of time. Using a scheduler in this manner allows the developer of a mission model flexibility to define the *coarseness* of the updates to the Mission Model, for example to accurately predict spin rates for a Gyroscope, or to plot the trends of a Heater Temperature.

The state of a Mission Model is defined at any particular epoch by a *Model State Vector*: it is a collection of Attribute values and statuses which evolve with time according to the behaviour definitions of the Mission Model. The Model State Vector may be saved and loaded so Mission Models may be initialised and recorded.

The behaviour of Model Elements may be defined in a number of different ways:

- from templates: a basic Model Element includes a template function which is called by the scheduler at the appropriate epoch. The function may be redefined by an engineer in order to describe the behaviour of the Model Element. Although the element of coding has not been removed, the scope of the coding has been significantly reduced: the template function performs all necessary actions, all that is required is for the behaviour of the Model Element’s attributes to be defined, for example how a Heater’s Temperature attribute changes with time. The software required to describe this behaviour refers to the attribute names so that the code uses natural domain language. For example, a developer of a Thermistor Model Element wants to code how the output voltage of the sensor relates to the input temperature. The developer creates an OutputVoltage attribute which is then related to an input value propagated by a thermal relationship. The name of this variable need not be known. The developer then simply writes code in terms of the OutputVoltage value and does not have to introduce variable names.

- predefined behaviours: a Mission Model includes a number of pre-defined behaviours, for example how to propagate a status change between a power source and a power sink. This behaviour is “attached” to the Model Elements by the Editor invisibly to the user. The Editor abstracts the user from much of the tedium of defining behaviours.
- interpreted statements: the Editor supports the user in writing behaviour by allowing the user to write code in a simple syntax which is then parsed and converted into executable code. This code is used to define Mode States, for example that a `Heater` is `TooHot` when its `Temperature` attribute is above a certain threshold. The user defines the statement; the editor generates the “if..then” code statement required.

Once defined for the Model Element class, all instances of the Model Element inherit the generic behaviours: the definitions need only to be made once and then they are reused.

Model Elements need also to react to commands, for example to define what happens when a `TapeRecorder` is commanded to play. This behaviour is defined generically within each Model Element so that the Model Element may be developed without prior knowledge of the actual command names used by each particular mission. This means that the genericity of each Model Element is preserved, the user of a Model Element simply maps the generic command effect to a real command name used in that mission. The use of the command flow relationship allows for complex modelling of commands, including command parameters, command processing and command routing.

The Model Elements may also respond to external events. These are events which are important to a Model Element but determining when they occur is outside the scope of the Mission Model. Examples of these type of events include invocation of failure modes, or spacecraft external phenomena such as eclipses or loss of signal. Similarly to command effects the definition of the effect of the event is generic: a user maps the generic event effect to the specific event effect name used by that mission.

3. THE MISSION MODEL EDITOR

A major output of the study is a Mission Model editor which is used by developers of a Mission Model. The Editor contains a palette which is used to contain generic Model Elements which may then be instantiated and used in the Mission Model. Each instance of the Model Element is then tailored to the needs of the mission, for example to specify the actual command name that turn a switch on, or the actual threshold levels that cause a `Heater` to become `TooHot`. A picture of the MBMCS Mission Model Editor is shown in figure 2.

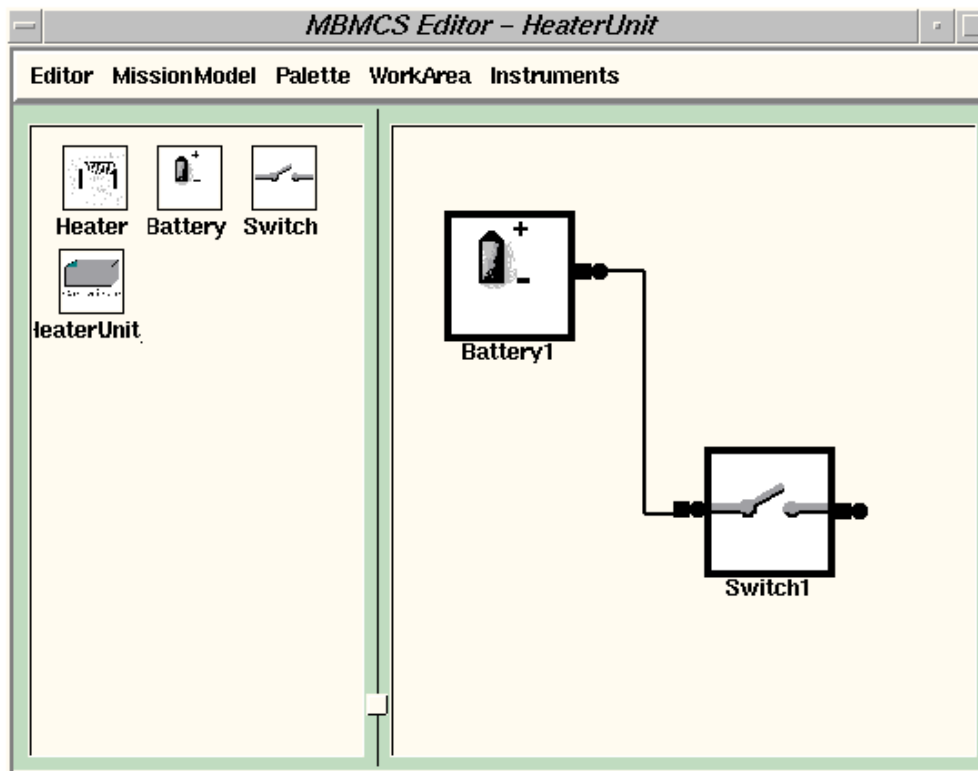


Figure 2: The MBMCS Mission Model Editor

During the process of creating a Mission Model, the user is also defining a graphical representation of the model. This graphical representation is used for display purposes and reflects the underlying state of the Mission Model. The display is configurable, for example colour changes may be used to reflect that the Heater is in ModeState TooHot, or that its powerflow relationship is inactive.

3.1 The Editor HCI

The HCI of the Editor provides a friendly interface isolating the user from the underlying Mission Model. The Editor allows a user to construct a model using graphic representations of Model Elements which may be physical such as a heater, a gyroscope or the entire attitude and orbit control system (AOCS), or logical, for example the onboard software. The user is able to use a library of Model Elements from which new ones may be specialised for the particular mission improving the amount of re-use within an MCS, hence saving time and reducing cost.

A user creates a Mission Model in a hierarchical manner: the user may define that a Gyropack contains six Gyroscopes and then use the Gyropack in the construction of the AOCS. The user lifts Model Elements from a palette and places them in a workarea in which the relationships between, and the behaviour of, Model Elements is defined. The user also configures the Model Elements in the Workarea, for example to specify the mapping between the generic command effects and the specific command names of the mission.

Relationships between Model Elements are also defined graphically. A line is drawn between two terminals by pointing and clicking the mouse, for example drawing a line representing a dataflow relationship between an Antenna and a Receiver. The Editor interprets this and updates the underlying Mission Model by creating a new relationship object and associating it with the Antenna and Receiver objects.

The behaviour of the Model may be defined in many different ways, in a full MBMCS implementation the Editor would provide support for a high level language allowing the user to define complex behaviour with intelligent support from the Editor. This support might include context sensitive editing so that the user is presented with an allowed set of keywords to choose from depending on the preceding statements, syntax checking and consistency checking. This high level language would be compiled into a suitable form for execution and extraction by an application using the Model.

4. The Future

The MBMCS study has successfully developed and implemented many of the modelling concepts introduced by the SCOS-II programme. These concepts are crucial to the success of a Model Based Mission Control System like that being prototyped by the ATOS-4 study.

The study has investigated standards for importing mission data, for example satellite manufacturers' engineering information, [MBMCS.95]. Importing such information directly into an editor would greatly ease the process of creating Mission Models. Of particular interest are the STEP and CALS initiatives from within industry for product data exchange; the use of such standards is being investigated by the ATOS programme [ATOS-4.96].

The ATOS-4 study is using the MBMCS Mission Model editor and model architecture to develop a mission model of ERS-2 for use during extensive trials which will take place in the first half of 1997. These trials will shadow real operations and be performed by operations staff; thus generating feedback from genuine end-users.

For an operational MBMCS, a more sophisticated operational Editor would be required, for example to allow the import of satellite manufacturer's data to aid the development of Model Elements, and support a high level language interpreter for behaviour definitions, however the prototype Editor has successfully demonstrated that it is possible to allow engineers to graphically develop and maintain Mission Models for use in an MBMCS.

5. REFERENCES

[AMFESYS.93] G Theiss, R Dobiash. Final report on ESA Study on Application of System Modelling in Spacecraft Control. (ref 8790/90/D)

[SCOS-II.94] N Head. SCOS-II - An Extensible Infrastructure for Mission Control. ESA PFF Vol 4 No 2.

[ATOS-4.95] J Wheadon et al. ATOS-4: A Mission Control System for Advanced Technology Operations. Proc. AI&KBS Workshop, ESTEC 1995.

[MBMCS.95] K Bradbury. Survey of Design Information, Management Systems and Standards. (ref EC.21304/WP200)

[ATOS-4.96] I. Simmonds et. al. The Advanced Technology Operations System (ATOS) Programme. Proceedings of SpaceOps96.