Model-Based Development of Spacecraft Onboard Functions

Takahiro Yamada and Keiich Matsuzaki
Japan Aerospace Exploration Agency / Institute of Space and Astronautical Science
3-1-1 Yoshinodai Sagamihara 229-8519 JAPAN

Abstract. This paper presents a method of development of spacecraft onboard functions based on a functional model of spacecraft (FMS). This model provides a standard framework for designing the functions of onboard components. It also enables development of a standard database (called the Spacecraft Information Base or SIB) that stores functional information on any onboard component.

To monitor and control onboard components, we have developed a standard protocol called the Spacecraft Monitor and Control Protocol (SMCP). Since the specific information on the functions of any onboard component can be retrieved from the standard database, the design of SMCP is very simple. SMCP specifies types of messages used for monitor and control spacecraft onboard components, and the formats of the messages used for each onboard component are stored in SIB, too.

The functional and operational characteristics of onboard components can be specified, to some extent, by parameters used by FMS and SMCP, which are stored in SIB. This information can be used by any application program including spacecraft testing and operations systems. Generic application programs that can be used to the design, testing and operations of any component or any spacecraft can also be developed because they can retrieve the functional and operational information on specific components from SIB.

Furthermore, the part of the software installed in onboard components that is responsible for the functional and operational behavior can be automatically generated from the parameters of FMS and SMCP stored in SIB, as long as it is described with these parameters.

Introduction

Presently, the functions of spacecraft onboard components are designed almost independently for each component without using any standards. This prevents reuse of the design of onboard components and the systems that support their development (design support systems, testing systems, operations systems, etc.) from component to component or from spacecraft to spacecraft. Furthermore, there are no standards on how to describe the functional design and operational rules for onboard components. Therefore, each designer has to devise a way of describing the functional design and operational rules for his or her components, and people who use this information (spacecraft testers, spacecraft operators, etc.) have to learn how to read the description written by each designer.

In order to solve these problems, this paper proposes a method of development of spacecraft onboard functions by using a functional model of spacecraft (FMS). FMS provides a standard way of designing the functions of onboard components. The FMS is based on the object-oriented

software technology and uses the concept of Functional Object (FO). FO is a unit for the functions of onboard components and captures information necessary for operating onboard components. By using this model, the functional design of onboard components can be specified in a formal and unambiguous way. Furthermore, reuse of functional design of onboard components will be easier because the information on components can be shared more easily among designers of onboard components. This will reduce the cost of developing onboard components.

By using a formal method for describing functional objects, a standard way of describing the characteristics of Functional Objects can also be obtained. FMS enables development of a standard database that stores the functional information of any components by using a standard way of describing the characteristics of Functional Objects. This standard database is called the Spacecraft Information Base (SIB).

To monitor and control onboard components from the spacecraft control system on the ground or from a computer installed on the spacecraft, messages (called command messages) are sent to the components and messages (called telemetry messages) are received from them. The Spacecraft Monitor and Control Protocol (SMCP) specifies the types and formats of command and telemetry messages and the sequences of their interactions. SMCP assumes that any onboard component is designed according to FMS. Therefore, it is designed to monitor and control Functional Objects. Since the characteristics of Functional Objects can be retrieved from SIB, the design of SMCP is very simple. SMCP can be used either between a spacecraft and its ground control system or between onboard components on the same or different spacecraft. The formats of command and telemetry messages for each specific onboard component are also stored in SIB.

By designing onboard components using FMS and SMCP, the functional and operational information of any component can be stored in SIB in a standard way as parameters used by FMS and SMCP, and this information can be used by any application program. This reduces the cost and risk of manual information transfer using documents.

Generic application programs that can be used to the design, testing or operations of any component can also be developed because they can retrieve the functional and operational information on specific components from SIB. This reduces the cost of developing supporting systems.

Furthermore, the part of the software installed in onboard components that is responsible for functional and operational behavior can be automatically generated from the parameters of FMS and SMCP stored in SIB, as long as it is described with these parameters. This reduces the cost of developing onboard components.

The basic elements of this method and their relationships are shown in Figure 1. In this figure, the arrows show the “developed from” relationship. For example, SMCP is developed from FMS.

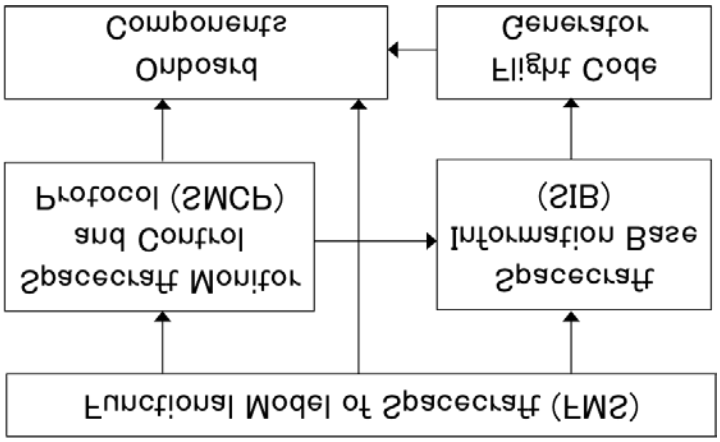

Figure 1. Basic elements and their relationships

Functional Model of Spacecraft (FMS)

Functional Model

The Functional Model of Spacecraft (FMS) provides a framework for designing the functions of onboard components. It does not specify ways of designing specific functions. For typical functions of spacecraft, we plan to develop a library of functional designs, which can be customized by component designers for their specific components. In this approach, what is meant by the functions of onboard components are an abstract representation of the jobs performed by the onboard components. They are abstract in the sense that only the aspects visible from the outside of the components are treated.

FMS was developed based on the Functional Viewpoint of the Reference Architecture for Space Data Systems (RASDS) (CCSDS 2008) and the Computational Viewpoint of the Reference Model of Open Distributed Processing (RM-ODP) (ISO 1996), but many extensions were added to these Viewpoints.

To design complex functions performed by onboard components, it is a common practice to decompose the functions into groups of functions, each consisting of a small set of functions closely inter-related to each other. In FMS, a group of functions defined in this way is called a Functional Object (FO).

Functional Objects

The concept of Functional Object is based on the concept of object used in the object-oriented software design methodology. However, there is a small difference between the concept of Functional Object used here and the concept of object used in the object-oriented software design. In the object-oriented software design, objects are dynamically instantiated from a class, while Functional Objects of this approach represent instances that exist permanently (although they may be active or inactive at specific times).

A Functional Object can contain other Functional Objects. In this model, a spacecraft is also a Functional Object and it contains all the other Functional Objects associated with that spacecraft. Therefore, Functional Objects are organized in a hierarchical way and the spacecraft is represented

by the root Functional Object. A Functional Object can be either active or inactive at any time and whether it is active or inactive is determined by the state (which will be explained later) that the parent Functional Object is in at that time.

A Functional Object is specified with the following concepts:

- Attributes,
- Operations,
- Alerts,
- Behavior, and
- Diagnosis rules.

An attribute is a parameter that represents the status of a certain part or a certain aspect of the Functional Object. The values of some attributes are analog (such as a temperature) and the values of some attributes are discrete (like an on-off status). An attribute may have a complex data type such as an array or a record. The value of some attributes can be set from the outside (through operations) but the value of some attributes cannot be set.

An operation is a function performed by the Functional Object and is invoked by receiving a command message from the outside. Operations include those for setting values to attributes. As a result of performing an operation, the values of one or more attributes usually change.

An alert is a function to report to the outside of the Functional Object that an important event has occurred inside the Functional Object. Alerts are delivered to the outside using telemetry messages. The occurrence of an event can be detected by the outside of the Functional Object by a change in the value of an attribute, but alerts are used to actively report the occurrence of important events. In a way, alerts can be considered to be the inverse of operations.

The behavior of the Functional Object is represented with one or more state transition diagrams. State transitions are triggered either as a result of performing an operation or by the occurrence of an internal event.

A state diagram is either active or inactive at any time. If the Functional Object itself is inactive, all the state diagrams associated with the Functional Object are inactive. Some diagrams are inactive while the Functional Object is active if the Functional Object is in a certain state (or in one of a certain set of states) of another diagram at that time. If a Functional Object has three state transition diagrams and is active at some time, it has one, two, or three active state transition diagrams. If the Functional Object has two active state transition diagrams at some time, it is in one of the states of one diagram and at the same time it is in one of the states in the other diagram. Therefore, it is in two states, each belonging to one of the active diagrams.

In what state of a state transition diagram the Functional Object is in at a certain time is indicated by the value of an attribute associated with that diagram. When a state transition occurs, the value of that attribute also changes. The values of some other attributes may change as a result of the transition. Each state of a Functional Object determines the set of operations that can be invoked when the Functional Object is in that state. Therefore, states specify what can be performed by the Functional Object at that time and the state transition diagrams specify the correct sequences of operations that can be performed by the Functional Object and how to verify whether each operation has been performed correctly.

Diagnosis rules specify rules used by the outside to determine whether a set of Functional Objects is in a dangerous condition. A diagnosis rule is defined by a logical expression performed against

the values of a set of attributes (this set can contain attributes from multiple Functional Objects).

The characteristics (that is, attributes, operations, etc.) of specific Functional Objects are stored in the Spacecraft Information Base (SIB) with a standard format.

Example of a Functional Object

The concept of Functional Object is illustrated with a very simple Functional Object. Let's suppose that a spacecraft called X has a Functional Object called A. It is used to represent the function of a component of this spacecraft. Functional Object A contains two other Functional Objects A1 and A2 (see Figure 2). Functional Object A is used to represent the general functions related to this component and Functional Objects A1 and A2 are used to represent some specific functions of this component.

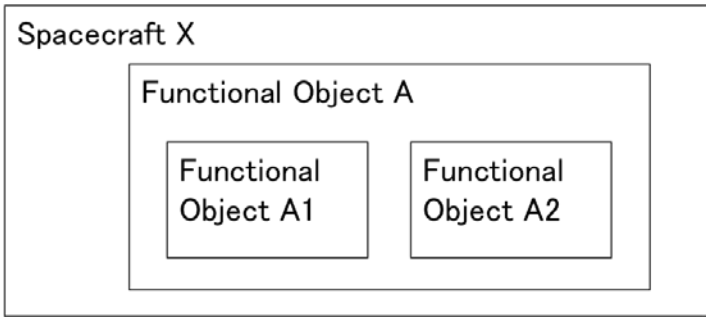

Figure 2. Example of Functional Objects

This Functional Object has the following attributes:

- A_OnOff,
- A_RunStop,
- A_ErrorStatus,
- A_CheckMode.

Attribute A_OnOff indicates whether this component is on or off. Attribute A_RunStop indicates whether this component is running or not. Attribute X_A_ErrorStatus shows the name of the error it has detected most recently. Attribute A_CheckMode shows what check mode it is using. The values of the first two attributes correspond to the states of the two state transition diagrams that this Functional Object has. Of these four attributes, only the value of A_CheckMode can be set by the outside by invoking an operation.

This Functional Object has the following operations:

- A_On,
- A_Off,
- A_Start,
- A_Stop,
- A_SetCheckMode.

Operations A_On and A_Off are used to turn on and off this component, respectively. Operations A_Start and A_Stop are used to start and stop the activities of this component, respectively.

A_SetCheckMode is used to set the value of attribute A_CheckMode.

This Functional Object has one alert, which is called A_ErrorDetected. It is used to report to the outside that some error has occurred. The kind of error that has occurred can be inspected by looking at the value of attribute A_ErrorStatus. The value of this attribute can also be attached to the alert message as a parameter.

The behavior of this Functional Object can be represented with two state transition diagrams (see Figure 3). Since the second diagram is active only when the Functional Object is in state On of the first diagram, it is shown contained in the first diagram. The labels attached to the arrows in the diagrams are the triggers of the transitions. Of these triggers, only A_ErrorOccurred corresponds to an internal event that is detected by the Functional Object. The other four triggers correspond to the operations performed by the Functional Object. When event A_ErrorOccurred is detected when the state is in Run, the state is automatically transitioned to state Stop and alert A_ErrorDetected is issued to the outside.

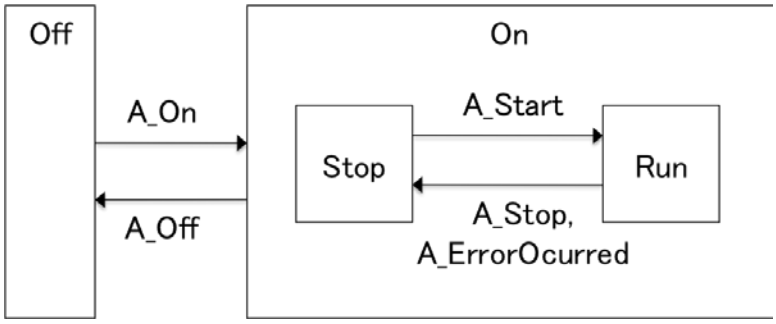

Figure 3. State transition diagrams of Functional Object A

The daughter Functional Objects A1 and A2 are active only when A is in state Run.

Spacecraft Monitor and Control Protocol (SMCP)

To monitor and control onboard components, messages (called command messages) are sent to the components and messages (called telemetry messages) are received from them. The Spacecraft Monitor and Control Protocol (SMCP) is an application layer protocol that defines types and formats of command and telemetry messages and sequences of interactions of messages to be used to monitor and control onboard components represented by Functional Objects (see Figure 4).

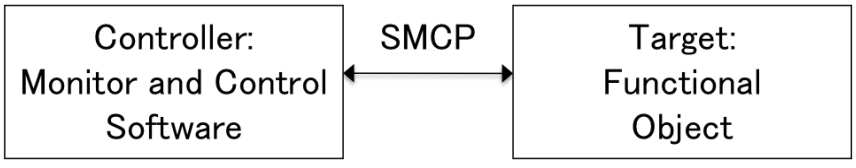

Figure 4. Spacecraft Monitor and Control Protocol (SMCP)

The entity that is monitored and controlled with this protocol is always a Functional Object or a set of Functional Objects representing onboard components, and they are called the Targets. The entity that monitors and controls the Target Functional Objects is called the Controller and is usually a piece of software that resides in a computer, which may be located either on the ground or onboard the same spacecraft or a different spacecraft. SMCP can be used between many different pairs of Controllers and Targets. For example, the {Controller, Target(s)} pair can be {a ground test or control system, an entire spacecraft}, {the spacecraft central data handling system, onboard subsystems}, or {a mission processor, science instruments}.

The Protocol Data Units (PDUs) of SMCP are typically transferred by the Space Packet Protocol (CCSDS, 2003), but they can be transferred by any network layer and/or transport layer protocol.

The SMCP also has the capability to adjust the volume of message exchanges depending on the available bandwidth of the link between the Controller and the Target.

The formats of command and telemetry messages used for a specific onboard component are stored in the Spacecraft Information Base (SIB) with a standard format.

The SMCP has the following two command message types:

- Action Command, and
- Get Command.

Action Commands are used for the Controller to invoke operations of a Target Functional Object. Get Commands are used for the Controller to trigger generation of Value Telemetry messages (see below).

The SMCP has the following three telemetry message types:

- Value Telemetry,
- Notification Telemetry, and
- Command Acknowledgment.

Value Telemetry is used for the Target to send the values of an attribute or a set of attributes. There are three ways for the Target to generate telemetry messages of this type. The first way is to generate instances of the same telemetry message periodically. The second way is to generate a telemetry message only when the Target has received a Get Command message to instruct the Target to send a telemetry message. The third way is to send a telemetry message when the value of the attribute has changed (or when the value(s) of one or more attribute(s) in the attribute set has/have changed). Notification telemetry is used for the Target to send an alert issued by the Target Functional Object. Command Acknowledgment is used for the Target to acknowledge receipt of a command message.

If the bandwidth of the link between the Controller and the Target is sufficiently high, the Target will send Value Telemetry messages at a constant interval. If the bandwidth of the link between the Controller and the Target is not sufficiently high, however, the Target will send Value Telemetry messages only when it has received Get Command messages and when there have been changes in the values of attributes. (In this case, loss of telemetry messages cannot be tolerated.) Therefore, it is possible to adjust the telemetry transmission method to some extent depending on the characteristics of the link.

Spacecraft Information Base (SIB)

The Spacecraft Information Base (SIB) is a standard database that stores (1) the characteristics of the Functional Objects developed for a spacecraft and (2) the formats of command and telemetry messages defined for that spacecraft according to SMCP.

The SIB contains such information as:

- Characteristics of Functional Objects,
  - Attribute names, attribute data types,
  - Operation names, operation parameter names, operation parameter data types,
  - Alert names, alert parameter names, alert parameter data types,
  - States, attribute values for each state, state transitions, trigger for each transition,
  - Diagnosis rules,
- Telemetry message formats, and
- Command message formats.

The contents of SIB are generated by the designer of the onboard components and maintained throughout the lifetime of the spacecraft (that is, during design, component testing, spacecraft integration, flight operations, etc). We have also developed an Excel-based tool to assist generation of the contents of SIB (see Figure 5). The contents of SIB generated with this tool are automatically converted to an XML document, which can be used by any application program, including the code generator explained in the next section. We will further develop another tool for generating SIB contents using graphical interfaces.

functionalObject definition 0 1 2 3 4 5 6 7 8 9	kind	kind property
Nozomi	functionalObject	
ScienceInstruments	functionalObject	
PWA	functionalObject	
CPU_Status	stateAttribute	**stateMachine** stateMachineCPU_Status
CPU_RunStatus	stateAttribute	**steteMachine** stateMachineCPU_RunStatus
HTR_Status	attribute	Enumeration {Ena, Dis}
ECC_Status	attribute	Enumeration {Ena, Dis}
ECC_1BitErrStatus	attribute	Enumeration {Error, NoError}
ECC_2BitErrStatus	attribute	Enumeration {Error, NoError}
WDT_Status	attribute	Enumeration {ENA, DIS}
WDT_ErrStatus	attribute	Enumeration {Error, NoError}
PWA_PacketFormat	attribute	Enumeration {1, 2, 3, 4}
CPU_ON	operation	
CPU_OFF	operation	
CPU_RUN	operation	
CPU_RST	operation	
HTR_ENA	operation	**attributeChangeRule** changeRuleHTR_StatusEna
HTR_DIS	operation	**attributeChangeRule** changeRuleHTR_StatusDis
ECC_ENA	operation	**attributeChangeRule** changeRuleHTR_StatusEna

Figure 5. Example of an Excel sheet to generate contents of SIB

Development of Onboard Software

The Institute of Space and Astronautical Science of Japan Aerospace Exploration Agency (JAXA/ISAS) is developing a standard onboard data handling architecture (Yamada et al. 2008), which will be used for all future science spacecraft of JAXA/ISAS. This architecture specifies a

stack of standard communications protocols to be used for communications between onboard components. These standard protocols are (from bottom to top of the stack) SpaceWire (ECSS 2003), the Space Packet Protocol (CCSDS 2003), and the Spacecraft Monitor and Control Protocol (this paper).

Software used for onboard components will also be developed based on this architecture. The layered organization of onboard software is shown in Figure 6.

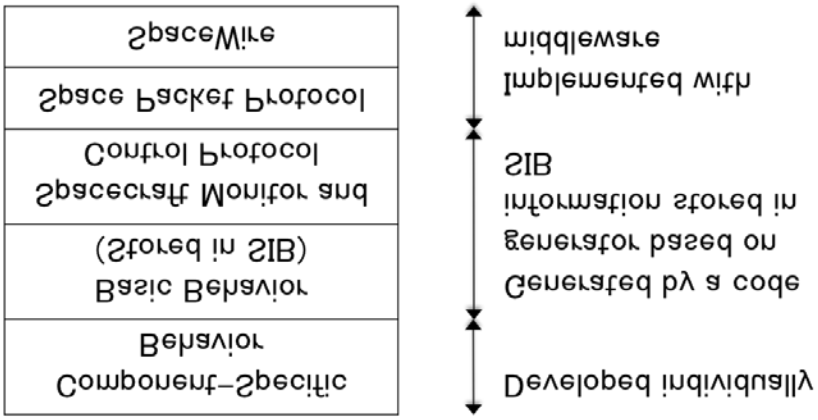

Figure 6. Layered organization of onboard software

The software for processing the SpaceWire protocol and the Space Packet Protocol is developed as middleware, which can be used by any onboard component.

The software for processing the Spacecraft Monitor and Control Protocol and for the basic behavior stored as state transition diagrams in SIB will be generated automatically by a code generator. Since the necessary information for each specific onboard component can be retrieved from SIB in a standard way, it is possible to develop a code generator that generates these layers of onboard software for any onboard component.

The software that implements component-specific behavior that cannot be expressed by state diagrams (such as attitude control logic) will be developed individually.

Conclusion

In this paper, we have shown how the spacecraft onboard components can be developed based on the Functional Model of Spacecraft (FMS) and the Spacecraft Monitor and Control Protocol (SMCP).

The cost of developing spacecraft can be greatly reduced with this model-based method because:

1) Designers of onboard components can use the standard design and description methods, and do not have to develop or learn different design and description methods;

2) Reuse of designs of onboard components becomes easier:

3) Generic application programs that can be applied to any component can be developed by

using the information on specific components contained in SIB;

4) Some part of flight software can be generated automatically by using the information stored in SIB; and

5) Manual exchanges of documents can be reduced by using SIB.

JAXA/ISAS is presently developing spacecraft onboard components based on these concepts for ASTRO-H (an X-ray telescope mission) and SPRINT-A (a EUV telescope mission), and will apply these concepts to all future science missions.

Acknowledgments

The authors would like to thank Dr. Masanobu Ozaki, Dr. Takeshi Takashima, and Dr. Tadayuki Takahashi, for many useful discussions on the concepts presented in this paper and the applicability of these concepts to future science projects of JAXA/ISAS. They would also like to thank Mr. Yusuke Iitsuka of Fujitsu for developing the Excel-based SIB generation tool mentioned in the paper.

References

Consultative Committee for Space Data Systems (CCSDS). 2003. Space packet protocol. CCSDS 133.0-B-1.

———. 2008. Reference architecture for space data systems. CCSDS 311.0-M-1.

European Cooperation for Space Standardization (ECSS). 2003. SpaceWire – links, nodes, routers and networks. ECSS-E50-12A.

International Organization for Standardization (ISO). 1996. Information technology - open distributed processing - reference model: architecture. ISO/IEC 10746-3.

Yamada T. and T. Takahashi. 2008. Standard onboard data handling architecture based on SpaceWire. In Proceedings of the International SpaceWire Conference 2008.

BIOGRAPHY

Dr. Takahiro Yamada (tyamada@pub.isas.jaxa.jp) is a professor at the Institute of Space and Astronautical Science (ISAS) of Japan Aerospace Exploration Agency (JAXA). He has been responsible for development of both onboard and ground communications and data handling systems for several science missions of JAXA/ISAS. He has also been actively involved in the standardization activities of the Consultative Committee for Space Data Systems (CCSDS) and edited more than a dozen standards published by CCSDS, which include standards on communication protocols and system architectures. He currently chairs the Space Communication Cross Support Architecture Working Group of CCSDS. He received the B.S., M.S. and Ph.D. degrees from the University of Tokyo, all in electrical engineering.

Dr. Keiich Matsuzaki (matuzaki@solar.isas.jaxa.jp) is an associate professor at the Institute of Space and Astronautical Science (ISAS) of Japan Aerospace Exploration Agency (JAXA). He has been responsible for development of the onboard mission data handling system and the ground data analysis and archiving systems for a solar telescope mission of JAXA/ISAS called Hinode (or SOLAR-B). He is currently involved in the concept definition of the next solar telescope mission of ISAS, and also leads development of a model-based software development environment to be

used for development of flight software of future science spacecraft of ISAS. He received the B.S., M.S. and Ph.D. degrees from the University of Tokyo, all in physics.