Automatic Generation of On-Board Software from the Model - Spacecraft Information Base Version 2 (SIB2)

By Keiichi MATSUZAKI1), Tetsu SAI TO2), Shinya OKUNISHI3) and Takashi KOMINATO4)

1) The Institute of Space and Astronautical Science, JAXA, Sagamihara, Japan
2) NEC Aerospace Systems, Ltd., Fuchu, Japan
3) NEC System Technologies, Ltd., Kobe, Japan
4) NEC Corporation, Fuchu, Japan
(Received June 27th, 2011)

In order to shorten time span for spacecrafts' onboard software, we have developed a Model Driven Architecture (MDA) tool named SIB2 generator: integrated development environment for onboard software. To SIB2 generator implemented is not only automatic source code generation but miscellaneous functions which accelerate implementation of application software. We also paid deep attention to enhance adaptability of the automatically generated code into various onboard components. This paper reports development-completion and design rational and results for initial version SIB2 generator.

Key Words: Onboard Software, Development Process, Modeling, MDA

1. Introduction

For decades, time span for requirement definition, implementation and verification of onboard software of space onboard components became much longer than the previous since requirements for onboard software became more complex. Now, the time span is significant in the development of whole spacecraft's mission. Thus, reduction of the steps in the software development is now key issue for recent spacecrafts' developments.

In general, there are two approaches to shorten time span to develop software: reuse of source code and adoption of model driven architecture (MDA; i.e. utilization of model throughout design, manufacturing and verification phases). While the former approach is conventional in the history of software development, the latter is less experienced and ambitious. We intend that both approaches are appropriately included into development of ISAS's onboard components. Since reuse of source code exceeds scope of a spacecraft's project, activity covering multiple projects is needed.

Most solid portion of software reuse should be implemented into middleware. At this moment, discussion for definition of middleware for onboard components developed by the Institutes of Astronomical in Aeronautical Science of Japan Aerospace Exploration Agency (ISAS/JAXA) is ongoing. Although there are some prototypes of middleware which are embedded into ISAS onboard components, implementation of these middleware has dependency on some specific hardware (e.g. Space Cube II 1-3) even if the hardware itself is intended to support variety of onboard components.

In a few years ago, MDA tools ready for spacecrafts' onboard components was not available. Thus, we started development of a MDA tool (SIB2 generator) as key tool for development of onboard software4). This paper is report of completion of the development.

Primary function of SIB2 generator is automatic generation of onboard software. Moreover, we implemented more functions for software development in order to minimize users’ effort and maximize users’ merit. In this paper, we report what we implemented in to SIB2 generator, an integrated environment for onboard software development. Section 2 describes merit of adoption of MDA tools for software development of onboard components in general. Following sections describe features of SIB2 generator together with modeling activity at ISAS.

2. Merit of Adoption of MDA Tools

Table 1 shows how MDA tools change process of implementation of spacecrafts’ onboard software. Upper half of the table shows typical process at ISAS in the previous decade. This process consists of 6 activities among which 5 activities (Steps 2-6) require human actions per design parameter. Among the activities, verification steps (steps 4 and 6) are required since previous steps (steps 3 and 5) are realized by human actions. Lower half of the table shows process improved by adoption of MDA tools. This process only has 1 activity (step 2) which requires human actions per design parameter. Among the activities, verification steps (steps 4 and 6) are required since previous steps (steps 3 and 5) are realized by human actions. Lower half of the table shows process improved by adoption of MDA tools. This process only has 1 activity (step 2) which requires human actions per design parameter. Comparing with the previous process, human actions per design parameter in the step 3 are removed since source code is automatically generated by MDA tool. Human actions per design parameter in the step 5 are removed by adoption of the design model applicable from onboard component development to operation-system development. Verification steps (steps 4 and 6) are not necessary since previous steps (steps 3 and 5) are not realized by human actions.

Although adoption of model in specification is powerful,
models cannot cover whole portion of specification of application. Nevertheless, effect of removal of human actions in the activities (step 3-6) becomes significant if the number of design parameter described in model is large. In order to remove large amount of human action in the activities, definition of the model, which can pick up large amount of parameters, is the most important issue.

In the following section, we introduce model and MDA tools at ISAS (Section 3) and scope of automatic code generation of SIB2 generator (Section 4). Regarding the step 3 in Table 1, we describe automatically generated source code (Section 5) and the supporting functions of SIB2 generator for implementation of software (Section 6). Regarding the step 4 in Table 1, we describe verification to the automatically generated source code (Section 7). Finally, We summarizes status and future plan for SIB2 generator (Section 8).

<table>
<thead>
<tr>
<th>Table 1. Activities for implementation of onboard software.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(a) Process in the legacy style.</strong></td>
</tr>
<tr>
<td>Step 1: Design and write down the algorithm into design-document.</td>
</tr>
<tr>
<td>Step 2: Determine detail parameters and write down the parameters into design-sheet.</td>
</tr>
<tr>
<td>Step 3: Implement the algorithm and detail parameters by hand coding.</td>
</tr>
<tr>
<td>Step 4: Verify the source code by unit tests.</td>
</tr>
<tr>
<td><strong>(b) Process achieved by adoption of MDA tools.</strong></td>
</tr>
<tr>
<td>Step 1: Learn algorithm described in our manual. Modify algorithm if the algorithm is not enough.</td>
</tr>
<tr>
<td>Step 2: Determine detail parameters and write down the parameters into the standardized design-sheet.</td>
</tr>
<tr>
<td>Step 3: Execute automatic source generation. (Step 4: Skip verification of the source code.)</td>
</tr>
<tr>
<td>Step 5: Convert the detail parameter into the model which is required by operation-system.</td>
</tr>
<tr>
<td>Step 6: Verify the conversion of the model.</td>
</tr>
</tbody>
</table>

### 3. Model and MDA Tools at ISAS

MDA tools at ISAS have adopted model 5,6) - functional model of spacecrafts (FMS) and an application level protocol - Spacecraft Monitoring and Control Protocol (SMCP). Parameters of the model are stored in database named Satellite Information Base version 2 (henceforth SIB2). Here, SIB2 is improved version of SIB (hereafter SIB1) which has been adopted for ISAS spacecrafts since the Planet-B (Nozomi) satellite which was launched in 1998. Based upon heritage of SIB1, SIB2 covers the definition of telemetry and tele-commands items which consist of large number of design parameters and is suitable for input to MDA tools.

Although SIB1 was theoretically adoptable for development of both onboard components and operation-system, SIB1 was not applied for development of onboard components in the most cases. One reason for the unfortunate situation is that supporting tools for onboard components’ development for SIB1 was not available. The other reason for the situation is that the model itself is not powerful enough to describe onboard components’ design. Although SIB1 only covers the information definition (~format of telemetry and tele-commands), SIB2 covers both the functional definition (~how onboard component behave) and the information definition. Here, the functional definition of FMS is the similar with subset of the Unified Modeling Language (UML). Since UML has been widely applied for development software, the functional definition is easily understandable for developer of onboard software and also suitable to implement automatic code generation tool.

Although MDA tools at ISAS are designed to coexist with standard technologies – e.g. technology applied at ISAS: space-wire networks, the standard building block of digital hardware (Spacecube2) and real time operating systems and middle-wares for onboard software, MDA tools at ISAS do not depend on these technologies. Developer of onboard component can adopt any operating systems and middle-ware etc. to implement their own application.

MDA tools at ISAS consist of several components: User interface of SIB2 (SIB2UI) which supports Step 2 described in the Section 2; Integrated environment for software implementation which supports Step 3; Generic Spacecraft Test and Operations Software (GSTOS) which support Step 4 and Step 6. Here, the integrated environment for software implementation generates source code of onboard component from SIB2. Thus, we have named this tool as ‘SIB2 generator’. In the following portion of this paper, we describe what we implement on SIB2 generator and rational for the implementation.
the generated files. Here, ‘own coding area’ denotes area in the generated files where users perform hand coding.

SIB2 generator generate source code in the assumption that microprocessor has either little or big (not middle) endian and express negative integer with two’s complement. This pause effectively no restriction since the most of microprocessor for spacecrafts’ onboard instrument agrees with the assumption.

4. Scope of Automatic Generation by SIB2 Generator

Although MDA tools can minimize cost of development of individual onboard components, development of MDA tool also take some amount of cost. Thus, we have selected most cost effective portion from the SIB2 model applicable for the automatic generation according to discussion among experts for development of software of onboard component at ISAS (Table 2). The definition of the scope is done in parallel with implementation of SMCP in to standards for telemetry and command for ISAS satellites. Thus, definition of scope might be updated for future version of SIB2 generator including the decision of implementation of SMCP.

Table 2. Scope of automatic generation by SIB2 generator.

(a) Functional Model of Spacecraft.

<table>
<thead>
<tr>
<th>Functional Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Attribute, State Attribute</td>
</tr>
<tr>
<td>- Operation</td>
</tr>
<tr>
<td>- Parameter</td>
</tr>
<tr>
<td>- Alert</td>
</tr>
<tr>
<td>State Machine</td>
</tr>
<tr>
<td>- State</td>
</tr>
<tr>
<td>- Transition</td>
</tr>
<tr>
<td>- Trigger</td>
</tr>
<tr>
<td>Attribute Sequence</td>
</tr>
<tr>
<td>- Reference of attribute</td>
</tr>
<tr>
<td>- Reserve</td>
</tr>
<tr>
<td>- Block</td>
</tr>
<tr>
<td>Event</td>
</tr>
<tr>
<td>Condition</td>
</tr>
<tr>
<td>Enumeration</td>
</tr>
</tbody>
</table>

(b) Spacecraft Monitor & Control Protocol (SMCP).

<table>
<thead>
<tr>
<th>Telemetry (Value Telemetry &amp; Notification Telemetry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Encode of parameters (not packing of identifier)</td>
</tr>
<tr>
<td>Command (Action Command &amp; Get Command)</td>
</tr>
<tr>
<td>- Decode of identifier and extraction information for unpacking of parameters</td>
</tr>
</tbody>
</table>

5. Automatically Generated Source code

Quality assurance is one of the most important topics in the development of spacecraft on-board software. In order to support human activities of the quality assurance, SIB2 generator generates source code that can be easily readable by users. To achieve the readability, SIB2 generator generates information of command decode and telemetry encode as tables since number of commands and telemetry items is large for typical onboard components. List 1 shows a sample of table type generation: command table. Contrary, SIB2 generator generates specific logic for processing of event and state transition since these can be complex and the number of appearance of logic in source code is expected to be relatively small. List 2 shows a sample of logic type generation: event detection. Not only model dependent code mentioned above, SIB2 generator also generates model-independent fixed codes to use the model-dependent code.

Logic of the processing is described in the specification document of SIB2 generator. If user can use the generated source code without modification what user should do is just to refer the description of logic of processing in specification document of SIB2 generator from design document of user’s application. In Figure 2 shown is an example of description of logic: sequence diagram of command processing generated by SIB2 generator.

In some cases, information described in SIB2 is not enough to generate source code. For example, functional object can be implemented by either hardware or software. Original of the functional model does not have information to specify portion for automatic source code generation. Such information can be described in extension field of SIB2UI and handled by SIB2 generator.

In some specific application, replacement of logic of the generated code is inevitable. For such cases, SIB2 generator provides a function to replace template of the source code. The template is described in the format defined by an open-source library Velocity (JAVA-based template engine) and users can replace the template. Although interface for the function is not friendly, existence of this function is expected to encourage improvement of automatically generated source code since the improvement can only be done by engineer of spacecrafts’ onboard software without engineer who developed SIB2 generator.

6. Supporting Functions for Implementation of Software

SIB2 generator is implemented as plug-in for Eclipse (standard platform for software development) since the activity of the automatic generation is closely located at following activity of hand coding of application software. Figure 3 shows screenshot of usage of SIB2 generator.

Eclipse is ported into various operating systems. SIB2 generator is also designed to be portable to variable operating system. However, compilation and verification of SIB2 generator is done only for Microsoft Windows since most of tools (cross compiler, In-Circuit Emulator (ICE), SIB2UI, etc.) operate on Windows.

SIB2 generator generate source code with own coding area. User shall implement application specific logic in the own coding area. For example, SIB2 generator generates stub functions of operation and attribute-getter and user shall implement their content. Unfortunately, some of MDA tools which generate source code overwrite the own coding area at the timing of the generation. This behavior is troublesome for users since users are forced to repeat hand-coding many times. To prevent this, SIB2 generator provides a function to protect content in own coding area. By this function, the own coding area is preserved at the timing of the code generation although other portion is replaced by new information from SIB2.
Sequence diagram of Action command processing is shown at Figure 5 and Figure 6. Each figures correspond to command receive to command decoder and operation processing of Figure 3, respectively.

**Figure 5** Sequence diagram of Action command receive that are generated from SIB2 XML of function object definition.

**Figure 6** Sequence diagram of operation drive that are generated from SIB2 XML of function object definition.

Fig. 2. Sample of description of processing logic.

List 1. Sample of table type generation: command table.

```c
#include "s2g_ActCommandDatabases.h"
#include "s2g_decodeActCommandFunctions.h"
#include "s2g_define.h"
#include "s2g_operation_${FO_NAME:1}.h"

/* Structure for Action Command Table for each FO */
struct s2g_ActCommandTable s2g_${FO_NAME:1}_ActCmdTable[S2G_${FO_NAME:1}_ACT_CMD_TABLE_SIZE] = {
    [${OPE_IDVALUE:1}, ${OPE_IDLENGTH:1}, s2g_Ope__${FO_NAME:1}_${OPE:1}, 1]
};

/* Structure for Command database for all FOs */
struct s2g_ActCommandDatabase s2g_ActCmdDatabase[S2G_ACT_COMMAND_DATABASE_SIZE] = {
    { S2G_FO_ID_${FO_NAME:1}, s2g_${FO_NAME:1}_ActCmdTable, S2G_${FO_NAME:1}_ACT_CMD_TABLE_SIZE }
};
```
List 2. Sample of logic type generation: event detection.

```c
#include "s2g_event.h"
#include "s2g_attribute_${FO_NAME:1}_${ATTR:1-1}.h"
#include "s2g_attribute_${FO_NAME:1}_${ATTR:1-2}.h"
#include "s2g_enumeration_${FO_NAME:1}.h"

/* user_own_code_start: s2g_event_include */
/* TODO: include header files */
/* user_own_code_end */

/* proto type for function of logical not operator */
static s2g_uint8_t s2g_eventNot(const s2g_uint8_t expression);

/** *
 * @brief function of logical not operator
 *
 * @param expression parameter for unary operator (not)
 *
 * @return result of not operation for input parameter
 */
static s2g_uint8_t s2g_eventNot(const s2g_uint8_t expression)
{
    return (expression == S2G_EVENT_NO) ? (s2g_uint8_t) S2G_EVENT_YES : (s2g_uint8_t) S2G_EVENT_NO;
}

/** *
 * @brief ${FORE_NAME:1}_${EVENT:1} check occurrence of event
 *
 * @return status for event occurrence (S2G_EVENT_YES:occurred,S2G_EVENT_NO:not occured)
 *
 * @note SIB2Generator
 */
s2g_status_t s2g_chkEvent_${FORE_NAME:1}_${EVENT:1}( void ) {
    /* definition of variables for attributes */
    ${ATTR_BIT_DATATYPE:1-1} ${FO_NAME:1}_${ATTR:1-1};
    ${ATTR_BIT_DATATYPE:1-2} ${FO_NAME:1}_${ATTR:1-2};
    /* definition of variable for judgment */
    s2g_status_t judge;
    s2g_uint8_t cond[S2G_EVENT_COND_SIZE_${FORE_NAME:1}_${EVENT:1}];
    /* get attributes */
    ${FO_NAME:1}_${ATTR:1-1} = s2g_getAttribute_${FO_NAME:1}_${ATTR:1-1}();
    ${FO_NAME:1}_${ATTR:1-2} = s2g_getAttribute_${FO_NAME:1}_${ATTR:1-2}();
    /* calculate term for judgment */
    cond[0] = ((${FO_NAME:1}_${ATTR:1-1} ${COMP_OPE:1-1} ${FO_NAME:1}_${ENUM:1}_${ENUM_TEXT:1-1}) ?
                (s2g_uint8_t) S2G_EVENT_YES : (s2g_uint8_t) S2G_EVENT_NO);
                (s2g_uint8_t) S2G_EVENT_YES : (s2g_uint8_t) S2G_EVENT_NO);
    /* total judgement */
    judge = (s2g_status_t) ${conditionExpression};
    return( judge );
}
```
In each own coding area, SIB2 generator generate comment tagged by a word ‘TODO’ and description of action. These tags are parsed by the standard development plug-in for C-language (CDT) and used for generation of task list. User should remove the TODO comment when implementation of own coding area is completed in order to remove the action from the task list.

In some of own coding area, default code is created. This code is applicable to some of onboard components without modification. Table 3 shows types of own coding area where default code is created. If default code is acceptable, what user should do is only to remove the TODO comment. Default code can be replaced by users. For example, encoding of message parameter tabulated in Table 3 can be replaced by more sophisticated algorithm (multi-bit processing) implemented by contributors of SIB2 generator. This code is also available for users of SIB2 generator.

7. Verification for Automatically Generated Source code

We intend that generated source code can be used without verification. Table 4 shows analysis applied for generated source code. Result of the analysis is documented and can be provided with SIB2 generator. In the result, conformance of the source code to MISRA-C rule set, developed by Motor Industry Software Reliability Association, is given. Although generated source code does not conform to some of the rule set, rational why some code generated by SIB2 generator does not follow the rule set is described in the document. This static analysis is performed with assistance of a tool: PARASOFT C++ test. In order to preserve information of acceptance of non-conformance, directives for “PARASOFT C++ test” to prevent message are described and kept in the source code.

<table>
<thead>
<tr>
<th>Model</th>
<th>Default Implementation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>File internal static variable</td>
</tr>
<tr>
<td>Execution of operation</td>
<td>Prompt execution after decoding</td>
</tr>
<tr>
<td>Encoding of message parameter</td>
<td>Bit by bit brute force packing</td>
</tr>
</tbody>
</table>

8. Status and Future Plan for SIB2 Generator

Development of initial version of SIB2 generator are completed at early 2011 after 3 cycles of trial releases and distributed to the end users. Number of onboard components which use SIB2 generator is now accumulated to 7 from 3 spacecrafts (Astro-H, Sprint-A and one additional satellite). The components include attitude and control system, data
handling unit and mission data processor. We plan to revise the SIB2 generator according to user comment from these initial users, hopefully ~2014.

Acknowledgments

The author also thanks Mr. Takuya Takahara and Mr. Tadashi Tanabe of NEC Aerospace Systems for fruitful discussions of onboard software and processes for developing the software at the present and also in the future. The author would like to thank Mr. Takey Sano and Mr. Ken Koizumi of NEC Corporation for leading the discussions and the implementing MDA tools.

References