SpaceOps COMMITTEE QUARTERLY

SOCQ5 – 3Q2011

Back to Products

ARTICLE 3

V.20 September, 2011

Title

Promising results from the customization of the Commanding and Monitoring Frameworks of INPE Satellite Control

Author(s)

Luciana S. Cardoso (INPE), <luciana@dss.inpe.br>
Paulo E. Cardoso (INPE), <paulinho@dss.inpe.br>
Joaquim P. Barreto(INPE), <joaquim@dss.inpe.br>

Correspondent

Mauricio G. V. Ferreira (INPE), <mauricio@ccs.inpe.br>

Technical Coordinator and Leading Reviewer

Patrick ("Pat") Hogan (CSA), <patrick.hogan@asc-csa.gc.ca>

Invited Reviewer

Joachim Kehr (DLR), <JoachimKehr@aol.com>

Editorial Supervisor

Eduardo W. Bergamini (INPE), <e.w.bergamini@uol.com.br>

Editorial Support

Síntique R. dos Santos (INPE), <secretaria.rme@inpe.br>
Helen Joyce Aparecida (INPE), <secretaria.rme@inpe.br>

Website Editorial Support

Megan Scheidt (AIAA/SpaceOps), <MeganS@aiaa.org>

Other information

<www.inpe.br>

Promising results from the customization of the Commanding and Monitoring Frameworks of INPE Satellite Control

Luciana S. Cardoso, Paulo E. Cardoso , Joaquim P. Barreto

National Institute for Space Research – INPE, São José dos Campos, Brazil

The Commanding and Monitoring frameworks are two fundamental stones of the SATellite Control System (SATCS) architecture at the National Institute for Space Research (INPE). They consist of a set of integrated classes which provide a pre-defined base infrastructure to support the development of applications for the remote control and monitoring domains. These domains include controlling and monitoring functions for both satellites and ground stations. These frameworks permit creating applications that could achieve a high level of software reuse with a lower development cost. In order to reach these targets, the following concepts have been widely used in the design: design-patterns, component-base development, and metadata. Recently, some software products have been created using the two frameworks and the promising results from their developing processes show that the objectives of maximizing reuse and decreasing costs have been achieved. In a short time while using a small group of developers two software applications were developed for the team that was testing the engineering model of the fourth China Brazil Earth Research Satellite (CBERS3): TMTCAIT (kernel functions for telecommand and telemetry processing) and CBERS3AOCSTC (decoding of AOCS telecommand binary files). The application DSSCOP1CTX to study the use of the CCSDS COP1 protocol for sending telecommands through the CORTEX baseband equipment has also been developed. The use of the frameworks and metadata were not the only reasons, taking into account the small staff and short deadlines, for the success in implementing these products. The reuse of the testing infrastructure of the frameworks and its validation process were also decisive. They have been created to validate the frameworks and had an important role in decreasing the effort for validating these three products and proved to be another important dimension of reuse.

Introduction

The Commanding and Monitoring object oriented frameworks were developed based on the architectural templates and design solutions proposed by SATellite Control System (SATCS) [1]. They are a set of integrated classes that provides a pre-defined base infrastructure to support the development of applications for the remote control and monitoring domains. These domains include control and monitoring functions for both satellites and ground stations.

The design of these frameworks took into account the common and variable requirements that can occur among different missions in order to make it easier to create the monitoring & control applications required for each specific mission. Using design patterns brings flexibility to change the code to comply with new requirements, since the code is designed to accept changes, thus saving time and reducing future development costs. Breaking down the frameworks in to modules makes it easier to change a specific part of the problem domains because the change would be concentrated in a specific module. By using metadata it is possible to reduce drastically, or, in some cases, even to eliminate the need to adapt the application codes to comply with new requirements. These frameworks allow creating applications that could achieve a high level of software reuse with lower development costs. To reach these targets, the following concepts have been widely used in the design: design-patterns[3], component-base development[4], and metadata[5].

Promising results from the customization of these frameworks have been obtained by the Ground System Development Division (DSS) team as three software products were developed in a short time while using a small group of people showing that the objectives of maximizing reuse and decreasing costs have been achieved.

This paper presents, in section II, an overview of the logical architecture of the Commanding and Monitoring frameworks and also the design solutions adopted to achieve a high level on the reusability of their elements. Other advantages of using frameworks are the reuse of testing infrastructure and the validation process which had an important role in decreasing the effort to validate the created products. Section III shows the test process adopted. The fourth, fifth and sixth sections describe the software applications which were developed using the frameworks: TMTCAIT (functions kernel of telecommand and telemetry processing), CBERS3AOCSTC (decoder of CBERS3 AOCS telecommand binary files) and DSSCOP1CTX (tool for helping the study of CCSDS COP1 protocol). The conclusion of this work is presented in section VII.

I. Commanding and Monitoring Frameworks

This section presents the overview of Commanding and Monitoring frameworks[2] which were designed to comply with the operation requirements associated with the telemetry, telecommand and ground station monitoring & control functions.

The Commanding framework covers functions of preparing, validating, sending, verifying and logging control messages (commands). This applies to any commandable entity like satellite (telecommand) and ground station (remote command) for which command messages can be defined.

The Monitoring framework covers all monitoring functions in which all data, regardless of the source (telemetry or remote monitoring), can be extracted, calibrated, subjected to a range of monitoring checks, archived in a *Historic* database, and displayed. It is also able to play back and display the data stored in the *Historic* database.

Figure 1 presents the logical architecture of the two frameworks where their functions are organized in four layers: *Persistence*, *Archiving&Retrieving Services*, *Subsystem and UserView* and an independent component, *Support*, which is responsible for providing support services for the first three layers.

The *Persistence* layer includes two databases: mission *Historic* data and *Operation* data (mission and software parameters). Metadata was used in the design of the *Operation* database to increase the flexibility to comply with the specific data structures requirements of each mission.

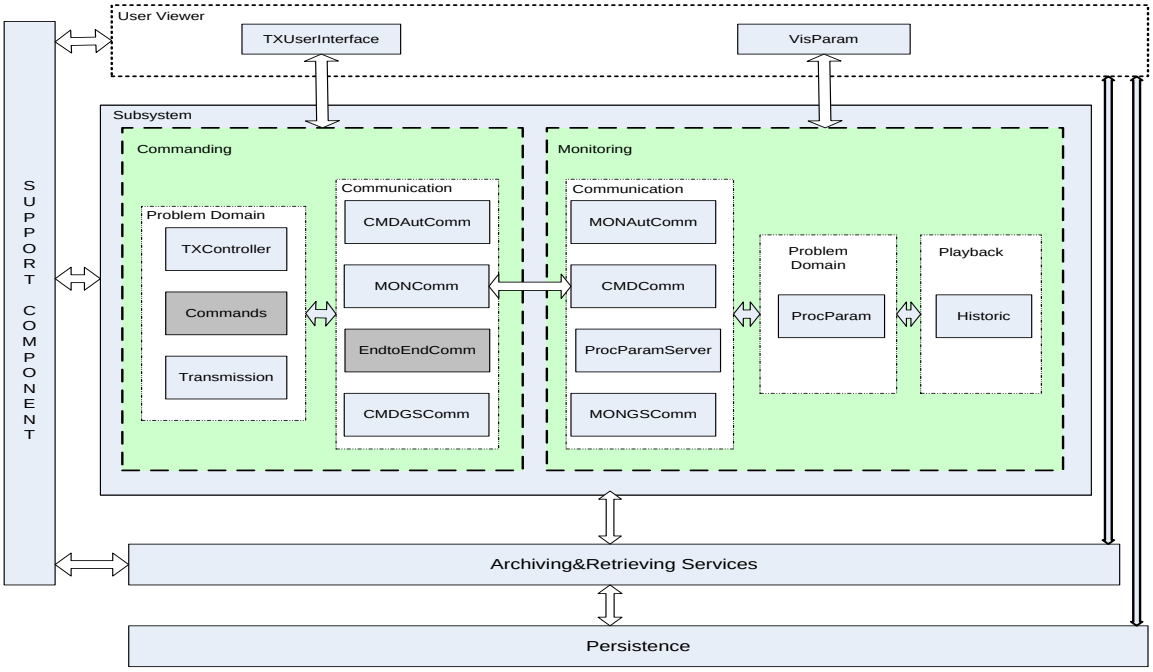

Figure 1. Commanding and Monitoring Framewoks Logical Architecture.

The *Archiving&Retrieving Services* layer makes it possible to design the subsystem independently from the implementation of the *Persistence* layer databases, by providing services for storing data and retrieving data from the *Operation* and *Historic* databases.

The functions related to the role of each subsystem, Commanding and Monitoring, are included in the *Subsystem* layer. Figure 1 also presents the logical modules that the problem domain of the two subsystems were brokendown in to.

The *UserView* layer includes only functions related to the interaction with the subsystem users: presentation of data managed by the *Subsystem* layer modules and activation, by the user, of services provided by these modules. The design of this layer produces one or more modules to comply with the subsystem user interface requirements and most of the time each module corresponds to one executable application with a graphical user interface.

Through the combination of these two frameworks, it is possible to create applications to control a remote entity in a manual, automatic or supervised mode. Requests for connection to a ground station, transmission of commands and others can be performed manually by an operator, through the application user interfaces (the *TXUserInterface* and *VisParam* modules), or automatically by the *CMDAutComm* and *MONAutComm* modules.

Whichever is the operation mode of the Commanding framework, all requests for transmitting commands are inserted in a queue of commands to be transmitted. The *TXController* module is responsible for controlling this queue and notifying the user interface whenever there is a change in the queue or in the execution state of a command. Depending on the operation mode, the commands are transmitted manually by an operator or automatically. The *Transmission* module controls the command transmission, by checking the range values of possible monitoring parameters associated with each command (pre and post verification), and notifies the Monitoring framework, due to command execution, about changes in the expected state of a remote entity. The communication between Commanding framework and ground stations are performed by the *CMDGSComm* module.

On the other hand, monitoring packets sent by a satellite (telemetry), through a ground station, or sent directly by a ground station (ground station remote monitoring) are received by the *MONGSComm* module. This module allows connections to one or more ground stations, storing the received packets in the *Historic* database and broadcasting them to several computers. Depending on the computer where it is installed, the *MONGSComm* module also plays the role of a client that receives the packets that were broadcasted. The packets are processed by the *ProcParam* module, whose functions includes: obtaining all monitoring parameter values and checking these values using monitoring laws. The *ProcParam* module also makes available these values and lists of parameters with values out of normal ranges to the *VisParam* module. Besides presenting the current values and processing states of the monitoring parameters, the *VisParam* module is also in charge of presenting the command queue to the user. Other features of this module are that it can be configured to present both real time and playback data. The only difference is that in the playback mode the *ProcParam* module processes packets retrieved from the *Historic* database, through the *Historic* module services, and not real time packets received through the *MONGSComm* module. Other SATCS subsystems can interface the Monitoring framework through the *ProcParamServer* module to receive parameters values extracted from *Historic* data. The *ProcParamServer* module is implemented as a server that receives requests from the SATCS subsystems and uses the *ProcParam* and *Historic* modules to provide the data these subsystems have ask for.

All communication interfaces between the Commanding and Monitoring frameworks are placed in the *MONComm* and *CMDComm* modules. The Commanding framework: (i) sends information about expected change in the state of a remote entity, due to command executions, so that the Monitoring framework can reconfigure the monitoring laws of parameters associated with these commands; (ii) receives parameter current values and uses them to check the pre and post verification rules associated with the

transmission and execution of a command and; (iii) sends information about the command queue and its updates to allow the users to monitor it.

The *Commands* and *EndtoEndComm* modules were designed as independent components that can be used by any subsystem or system that needs to code/decode the application data of a command and to pack/unpack the protocol data layers that can exist to allow the communication with remote entities (ground-board protocol and/or ground node to ground node protocol).

The design of the *Commands*, *EndtoEndComm*, *CMDGSComm*, *MONGSComm* and *ProcParam* modules makes use of the *Data Stream* module[2] of the *Support* component to minimize the need of changing the code to comply with requirements of mission specific monitoring and command data structures.

An adequate configuration of these frameworks allows creating software products for different mission phases.

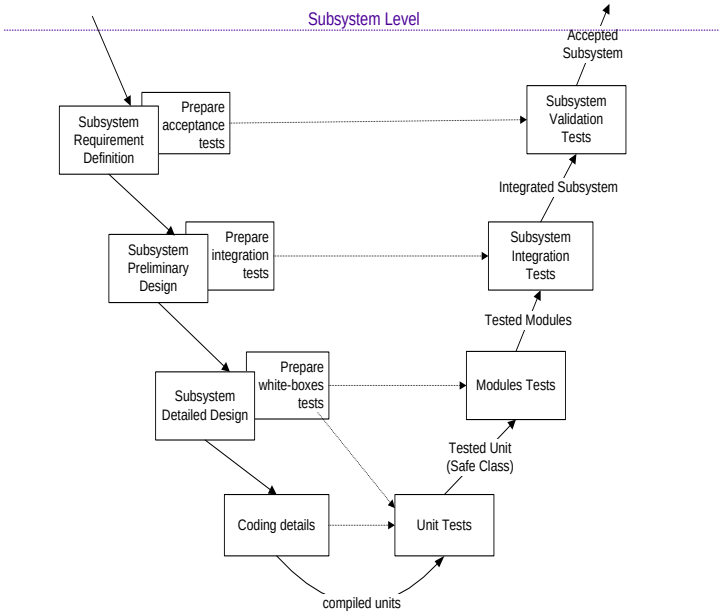

Figure 2. Framework Test Phase

II. Testing infrastructure and validation process

The test lifecycle adopted for validating the software products created from the framework customization is the V-Model as shown in Figure 2. The planning of the tests is performed at each phase of the development, in a top-down strategy. However, the test execution and result reports are applied and generated respectively, in a bottom-up fashion. The V&V process aims at achieving the maximum reuse of the testing tools and the adoption of cost effective techniques in order to assure the required quality of

software products. The activities of the V&V process run in parallel to all phases of the development process.

The adoption of different phases of tests in this process makes it easier, for each framework customization, to reuse all the test tools developed (like drivers, stubs and simulators) and all the test case sets prepared (procedures, input and output data). Thus, the work of preparing the tests for a new framework customization concentrates on the insertion of new test cases to validate the new requirements of the software module/product to be developed. However, for a complete validation of the customization it is necessary to apply not only the new test cases but also, in the case of modified modules, the whole test suit previously prepared (regression testing).

Another advantage of this validation process is that each module has a validation phase separated from the other modules and later on there is a validation phase to integrate all these modules. In this way, to validate a new software product that require changes on a subset of the frameworks modules, it is possible to test only the modified modules and then apply the integration test cases to the affected modules. To adapt themselves to this test process, the drivers developed for testing each module offer a means to execute all services made available to the clients by that module. These services are composed of the ones defined in the module interface class, implemented through the Façade design pattern, plus call back services for handling asynchronous events generated internally to that module, implemented by the Observer design pattern.

Another complex and important item of the testing infrastructure is the Operation database. Since the frameworks use metadata intensively, in other words, a major part of implementation depends on the data stored in the Operation database of each system, a strategy to mitigate the work of inputting test data into the database was also defined. Figure 3 illustrates this strategy. First, an Operation database instance is made available to be edited. This instance includes data required by the framework implementation (software configuration data) such as the set of event messages that the framework can generate during its processing and the type of data contained in a command. This kind of information is strictly connected to the existence of specific code in the framework model that uses it.

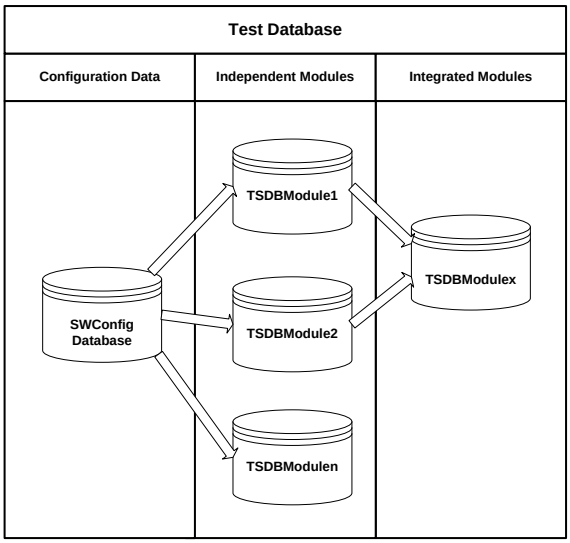

Figure 3. Testing database instances

The first instance of the database is replicated initially for each module of the subsystem that is independent from the other modules (leaf modules). Then a set of data is prepared for the specific tests of these modules. If implementations of new requirements are added to a module then the data to test this modification should be inserted into the instance of the database of that module to allow the execution of these new tests and reapplying the whole test plan of that module whenever it is necessary.

A module which makes use of the independent modules has its database instance created from the initial instance (configuration) joined to the database instances of the modules on which it depends. Besides that, whenever necessary, new data could be added. To support this database management process a tool was developed to copy the contents from one database instance to another, preserving their integrity rules.

The implementation of the first software product that allowed testing of the proposed framework architecture was accomplished with the third China Brazil Earth Research Satellite (CBERS2B) control and monitoring application (SATCS_TMTC_CBERS2B). This application was chosen because the CBERS2B satellite was already being controlled by another application and therefore it was possible to match the results generated from both control systems. Also considering that after the completion of the development there would be a basic TM/TC core already validated for new satellites from the CBERS family. The results obtained from the tests of this product were successfully compared with real data produced by the current CBERS2B control and monitoring system.

Having, as the starting point, the frameworks modules implemented and validated, the framework architecture was customized and validated by the first software product and the existing test infrastructure, it becomes much easier to create new frameworks customizations and validate these new products. Three software products, which are described in upcoming sections, were developed in a short time using a small group of

developers, showing that the objectives of maximizing reuse and decreasing costs has been successful.

III.TMTCAIT

The software product TMTCAIT is a kernel of functions for telemetry processing and command transmission that was developed with the aim to allow commonality between telemetry and telecommand systems used in satellite assembly and integration test (AIT) as well as flight operation phases. The purpose was to develop or use different graphic user interfaces for Operation and AIT systems but both using a single processing kernel and the same Operation database.

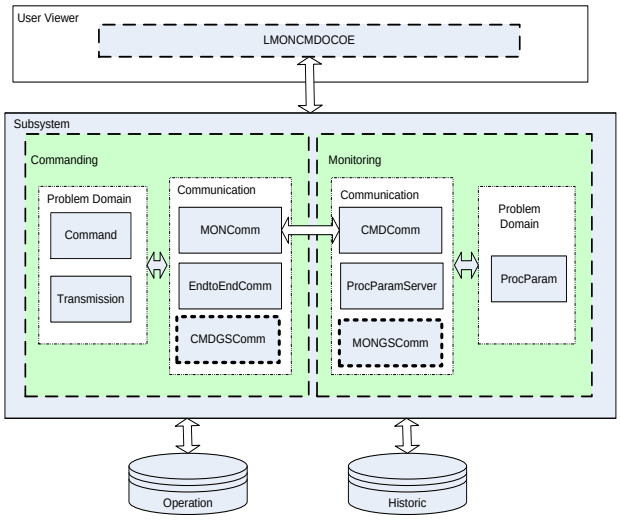

Figure 4. TMTCAIT Logical Architecture.

The TMTCAIT was created based on the customization of Commanding and Monitoring frameworks. Figure 4 shows the logical architecture of the product along with User View and Subsystem layers customized and the Operation and Historic databases. To integrate the AIT applications with the command and monitoring frameworks it was necessary to create a dynamic library (LMONCMDOCOE) using the C language. This interface was created because the frameworks were developed using the C++ language and the Microsoft Visual Studio 2008 IDE while the IDE of the AIT development team is Lab Windows. This library uses the following modules from the frameworks: Command, Transmission, MONComm, EndtoEndComm, CMDGSComm, CMDComm, ProcParamServer, MONGSComm and ProcParam.

The extra coding necessary to create the TMTCAIT product was the development of the LMONCMDOCOE library and changes to the CMDGSComm and MONGSComm modules. For these two modules, classes were added to support the TM and TC communication interface protocols of the baseband equipment used by the AIT team. Two classes of these modules, implemented according to the Factory Design Pattern,

were modified in such a way to permit creating the objects of these new communication classes.

To validate the first version of TMTCAIT, the Operation database was configured for the CBERS3 satellite. Figure 5 shows the test environment created to validate TMTCAIT. The driver GUIMONCMDOCOE was developed to test every function of the LMONCMDOCOE. Two simulators of the TM and TC interfaces of the AIT baseband equipment (SIMCMDLITGS e SIMMONLITGS) were created in order to allow testing in the development environment. In addition to these testing tools, two other applications that are part of SATCS were used. The OPDBEditor application, that allows editing the Operation database, and the PARConsultaDB application, used to query the Historic database. CBERS3 telemetry and telecommand parameters were inserted into the Operation database using the OPDBEditor application. The configuration of these parameters in the database were enough to process telemetry and code telecommands without any software modifications in the modules responsible for coding/decoding data structures. The module EndtoEndComm, that was modified to comply with SATCS_TMTC_CBERS2B product, was totally reused since this part of the CBERS3 ground-board protocol is the same as the CBERS2B.

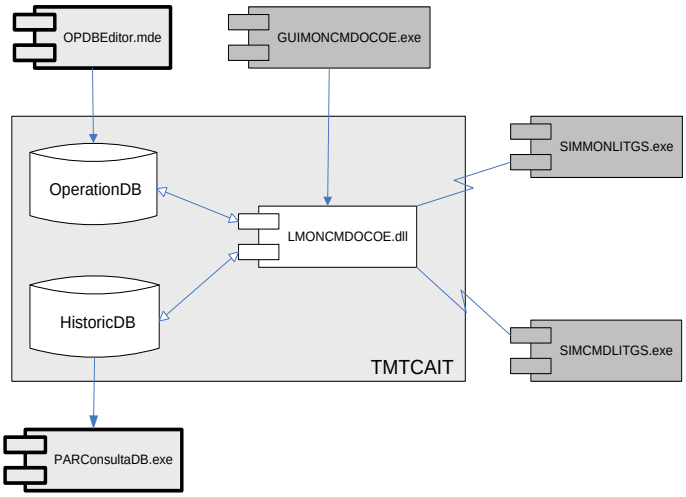

Figure 5 TMTCAIT Test Environment.

In April 2009, a first version of TMTCAIT was successfully tested in the AIT environment of the CBERS3 engineering model in China. Since the AIT team did not yet have software to access LMONCMDOCOE, the TMTCAIT test environment was taken to China where the TM and TC simulators were replaced by the actual equipment. Besides that, a telemetry viewer application developed for the validation of the Monitoring framework was used to visualize the telemetry values during the tests. The major adjustments that were made related to telemetry parameters in the database in order to correct faulty data. A small modification in the CMDGSComm module code was also performed due to a misunderstanding concerning the telecommand interface of the AIT baseband equipment.

At this point the DSS and AIT teams are reassessing the functions made available in the first version of LMONCMDOCOE in order to change some functions and include some others to better comply with the requirements of the AIT system.

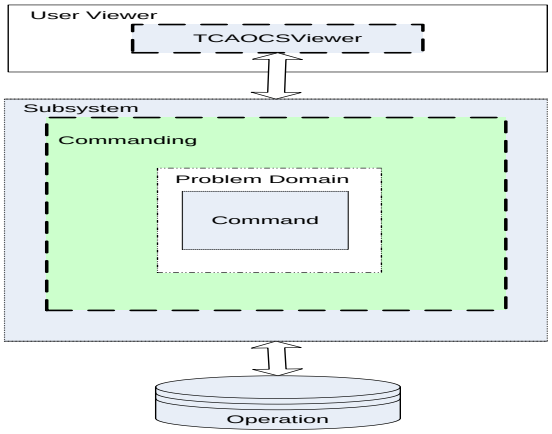

Figure 6. CBERS3AOCSTC Logical Architecture .

IV. CBERS3AOCSTC

The Commanding and Monitoring frameworks were also designed to allow the possibility of creating applications, by selecting a subset from the framework modules, in order to help satellite experts during the testing phase of their subsystems. This could also be verified during the CBERS3 engineering model tests in China. The development and testing of the satellite AOCS subsystem is under the Chinese team responsibility but Brazilian experts need to monitor it. Since AOCS command format was changing, the Brazilian team was spending considerable time manually decoding the binary files provided for testing and confirming their contents. Thus, it was realized that an application based on the frameworks could be created to decode the files in order to save them time.

The software product CBERS3AOCSTC was then created to help the AOCS subsystem experts. This application allows decoding binary files of AOCS commands and presenting the values of their parameters to the user. Figure 6 presents CBERS3AOCSTC logical architecture with its User Viewer and Subsystem layers and the Operation database. The User Viewer layer was implemented, through an user interface called TcAocsViewer which displays the decoded contents of the binary files and also allows the user to create reports. The subsystem layer, which is composed only of the Command module of the Commanding framework, is responsible for decoding the AOCS command. This tool was developed and tested in Brazil and sent to Brazilian experts in China, along with OPDBEditor tool to allow that the Brazilian experts could themselves change the Operation database whenever this was necessary.

The major work was concentrated on the Operation database configuration by inserting data to describe the structures and parameters that compose the AOCS telecommands. It took 33 man-hours to prepare the database and 10 man-hours to code the graphical interface.

Due to the speed at which the product was developed and the good results achieved by reducing the time to monitor the AOCS tests, other subsystems experts were interested in other applications that could be developed to help them in coding and decoding their data structures.

V. DSSCOP1CTX

Since the Amazonia satellite, in development by INPE, will use the CCSDS protocol and considering that the DSS team has no experience in this protocol, it was decided to develop a software product, to study it. In this study the Cortex-COP1 software and the CORTEX baseband equipment were used. The software product DSSCOP1CTX was developed for this purpose. It can be configured for both Cortex-COP1 remote controlling and monitoring (data control) and sending directives and telecommands (telecommand data flow).

The DSSCOP1CTX logical architecture, including its User Viewer layer, Operation and Historic databases, as well as the customized Subsystem layer, is presented in Figure 7. In the User Viewer layer a graphical user interface was implemented called COP1CTXViewer to allow the stimulation of the Cortex-COP1 communication interface protocol. The following frameworks modules were used: Command, Transmission, EndtoEndComm, CMDGSComm, ProcParamServer, MONGSComm and ProcParam. Modules, whose rectangles in Figure 7 are dotted, had to be modified to implement new requirements of the Cortex-COP1 communication interface and CCSDS COP1 protocols.

New classes to handle the telecommand data flow and the data control interfaces of the software Cortex-COP1 were added to CMDGSComm module. The class used to create the communication objects of this module, implemented according to the Factory design pattern, was modified in order to create the new objects of these communication classes. Asynchronous events generated by the Cortex-COP1 (Alert Indication and Additional Information) were managed by adding their handling to the class responsible to handle asynchronous events. The design of this event handling class uses the Abstract/Observer design pattern. This caused changes in the Transmission module because it is responsible to instantiate a concrete Observer class which inherits from the abstract observer class defined in the module CMDGSComm in order to handle these new events.

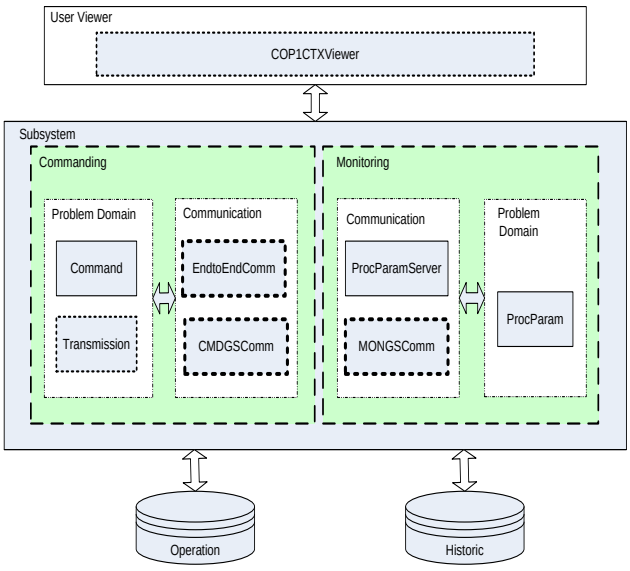

Figure 7. DSSCOP1CTX Logical Architecture.

Classes responsible for processing the CCSDS Packet and Segment layers as well as the classes to process communication protocol layers of the software Cortex-COP1 were added to the EndtoEndComm module.

In the MONGSComm module a class was added for implementing the client communication protocol interface of the monitoring port of the Cortex-COP1 software. In order to decode telemetry frame parameters, including CLCW parameters, and make it easier to monitor the tests, other classes were added to this module for implementing the client communication protocol interface of the telemetry port of the CORTEX TM.

An Operation database instance was created to describe data structures and parameters of Cortex-COP1 remote command&monitoring data messages, COP1 directives and telecommands.

The software infrastructure prepared for studying Cortex-COP1 software and CCSDS COP1 protocol is presented in Figure 8. Two instances of COP1CTXViewer were used to configure the tests. One instance (RMRCAPP) was used to monitor and control the Cortex-COP1 software. The other (TCAPP) was used to test the telecommand data flow interface of the Cortex-COP1 software. Both instances used the same Operation and Historic databases. Two simulators were developed to simulate the Cortex equipment. The SIMCMDCTXGS application simulates telecommand functions and the SIMMONCTXGS application simulates telemetry functions generating telemetry frames with CLCW for the Cortex-COP1 software. Two instances of the VisMonTReal application were also used. This application allows receiving, monitoring and visualizing remote monitoring parameters. One instance (RMAPP) was used to visualize Cortex-COP1 remote parameters and the other (TMAPP) to visualize telemetry parameters generated by SIMMONCTXGS. The last and most important element of the test environment was the Cortex-COP1 software that was the object under study. It interfaces with DSSCOP1CTX and the simulators. Besides the OPDBEditor and

PARConsultaDB applications, already described in chapter IV, were used to prepare the Operation database and query historic data stored during the tests.

The effort to implement and test the DSSCOP1CTX Data Control version was 113 man-hours, split into 85 man-hours for designing and implementation, and 28 man-hours for testing execution. To implement and test the DSSCOP1CTX Telecommand Data Flow version took 164 man-hours, where 99 man-hours were spent in designing and implementation, and 65 man-hours for testing execution. A set of test cases were elaborated for stimulating the Cortex-COP1 monitoring and control interface and also its command data stream port. These test cases aimed to verify all possible data messages exchanged with the Cortex-COP1 software and part of the CCSDS COP1 protocol.

If creating a tool for studying the Cortex-COP1 software was a quick and efficient process the credit has to be given to the good qualities provided by the Commanding and Monitoring frameworks and to some of their design characteristics: (i) it was possible to comply with most of the requirements by inserting data in the Operation database (metadata); (ii) the use of design patterns in the design of its modules makes it easy to change the code whenever it is necessary; (iii) and its testing infrastructure and its validation process reduce the effort for validating its customizations. In the next step to conclude these studies the simulators will be replaced by the Cortex equipment and the test case suit reapplied.

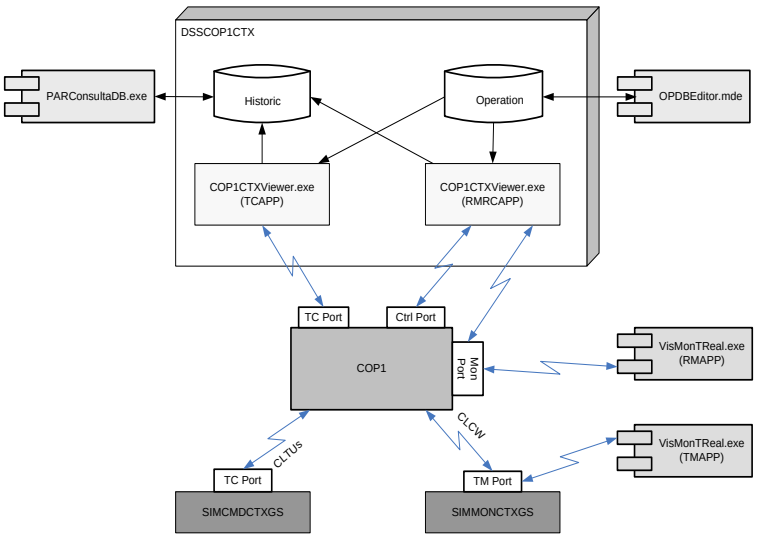

Figure 8 DSSCOP1CTX Sw Infrastructure.

VI. Conclusion

Writing this paper was very gratifying because the results obtained with the development of three software products (TMTCAIT, CBERS3AOCSTC and DSSCOP1CTX) demonstrated that the proposed goals outlined at the beginning of SATCS project have been achieved. There was a high level of software reuse and a lower cost of development.

The development and testing of TMTCAIT product proved that the commonality between AIT and flight operation phases is possible, at least through the use of a unique telemetry processing and telecommand transmission and a single Operation database. Therefore, at the end of the AIT phase, the Operation database can be simply transferred to the operation system without the need to edit all telemetry and telecommand parameters. This will save a lot of work for the operation team and the Operation database will be more reliable.

Because the CBERS3AOCSTC product aroused the interest of other satellite subsystem experts, one can foresee the use of frameworks to quickly and efficiently create small tools to help them. Since the major complexity to create these tools is related to the correct database preparation, an editor has been developed. It was designed to make it easier for the database to be updated by users.

The good results obtained from the DSSCOP1CTX development to study the CCSDS COP1 protocol and the Cortex-COP1 software proved that the logical architecture designed for the frameworks offers great flexibility to create applications for new communication protocols and new data structures without any major coding impacts. The changes are restricted to a small group of classes. Most of new requirements are met by the correct configuration of the Operation database.

The next software product to be developed will be a tool to help in the testing of the ACDH (AOCS + OBDH) subsystem that has been developed for the Amazônia satellite. This satellite will use the CCSDS protocol and the PUS (Packet Utilization Standard). This software will offer an interface to the test control system to allow telecommand transmission, and telemetry reception, processing and visualization. This software will be developed based on the TMTCAIT and DSSCOP1CTX products. The new elements are the implementation of the PUS protocol functions and of the interface to the test control system.

Acknowledgments

Authors would like to thank all the members of the DSS software team since without everyone's dedication the results presented in this paper would not have been achieved. Special thanks to Leandro Toss Hoffmann who had an important role in coding frameworks modules.

References

Cardoso, P. E.; Barreto, J. P.; Dobrowolski, K. M., "A Ground Control System for CBERS 3 and 4 Satellites," *9th International Conference on Space Operations*, AIAA, Rome, Italy, 2006

Cardoso, P. E., Barreto, J. P.; Cardoso, L. S., Hoffmann, L.T., "Using Design Patterns, Components and Metadata do design the Command and Monitoring Frameworks of the INPE's Satellite Control System", *10th International Conference on Space Operations*, AIAA, Heidelberg, Germain, 2008

Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed., Addison-Wesley, Reading, Massachusetts, 1995, pp. 395.

Sommervile, I., *Software Enginnering*, 6th ed., Addison-Wesley, Harlow, England, 2001, pp. 310-318.

Johnson, R.; Wolf, B., "Type object," *Pattern languages of program design 3*, edited by R. C. Martin, D. Riehle and F. Buschmann, Addison-Wesley, Reading, Massachusetts, 1998, cap. 4, p. 47-66.