

AN EXPERIENCE IN DESIGNING SOFTWARE FOR MICRO-SATELLITE CONTROL USING GRAPHICAL LANGUAGE

Maria F. MATTIELLO-FRANCISCO & Marcos S. GUIMARÃES & Manoel J. M. de CARVALHO

National Institute for Space Research -INPE

Av. dos Astronautas, 1758

12227-010 São José dos Campos, SP, Brazil

Fax (+55)(12)3456589 E-mail: {fatima, marcos}@dss.inpe.br, manoel@crn.inpe.br

ABSTRACT - *SACI-SOL software is the heart of Scientific Micro-satellite (SACI) Ground Station. It consists of the Telemetry and Telecommand (TM/TC) software developed to automate satellite control as much as possible, enabling SACI micro-satellite operations to be carried out by a minimum staff.*

In order to meet the low project cost requirements reducing development time and increasing productivity, LabVIEW graphical environment was adopted.

Focusing on that software development graphical language, this article presents an overview of both the SACI-SOL design solution adopted by the software development team and the way of implementing, in LabVIEW, classical solutions such as finite state automata, event control, queue mechanisms, among others.

Also, an analysis of TM software is done in order to evaluate its performance in receiving, recording, processing and displaying 60 packets/s (1Kbyte per packet).

1. INTRODUCTION

SACI-1 is a low earth polar orbit scientific micro-satellite intended for systematic observation of the ionosphere. It shall be launched as CBERS (China-Brazil Earth Resource Satellite) piggyback, by Chinese launcher Long March IV, on ending of 1998.

There are 4 scientific experiments on-board of SACI-1 which will acquire data from the space plasma environment autonomously and simultaneously during the entire mission life. The payload telemetry plus the satellite telemetry will be transmitted to Ground Station in 500 Kbps rate (an average of 60.4 packets per second).

Matching the satellite orbit with the SACI-1 Ground Station position (at northeast of Brazil), the satellite visibility shall occur twice a day, in 5-8 minutes passes. As a consequence of both the SACI short contact and the high data payload acquired by the on-board scientific experiments (~36 Mbytes per day), the Ground-Satellite protocol communication, PROSOL, shall be robustly implemented and the TM/TC application software shall provide essential autonomous features such as unattended control of a satellite passing.

In order to meet those requirements, SACI-SOL was designed to operate in two Personal Computers platforms, under Windows NTTM operation system, interconnected by serial link RS-232. One of them is dedicated to perform TC control while another is doing TM functions. The serial link between them is used for data transmission related to the protocol communication tasks. The figure1 shows the SACI-SOL architecture. It consists of the following software modules :

1. *TM Physical & Codification Module* – deals with the TM data acquisition, bit and word synchronization and frames decommutation.
2. *TM Transfer Module* – deals with the TM frames sequence validation following European Space Agency recommendation [ESA 90].
3. *TM Application Module* – deals with the high level functionality of storing and treating each TM packet received from satellite. That includes the identification of the TM packet type, the processing and the visualization of the relevant one.
4. *TC Physical & Codification Module* – deals with the codification and transmission of the TC bits to the satellite.
5. *TC Transfer Module* – deals with the treatment of the TC frames following European Space Agency recommendation [ESA 90].
6. *TC Application Module* – deals with the high level functionality of the flight plan automatic control in order to transmit all the telecommands selected for each passage.

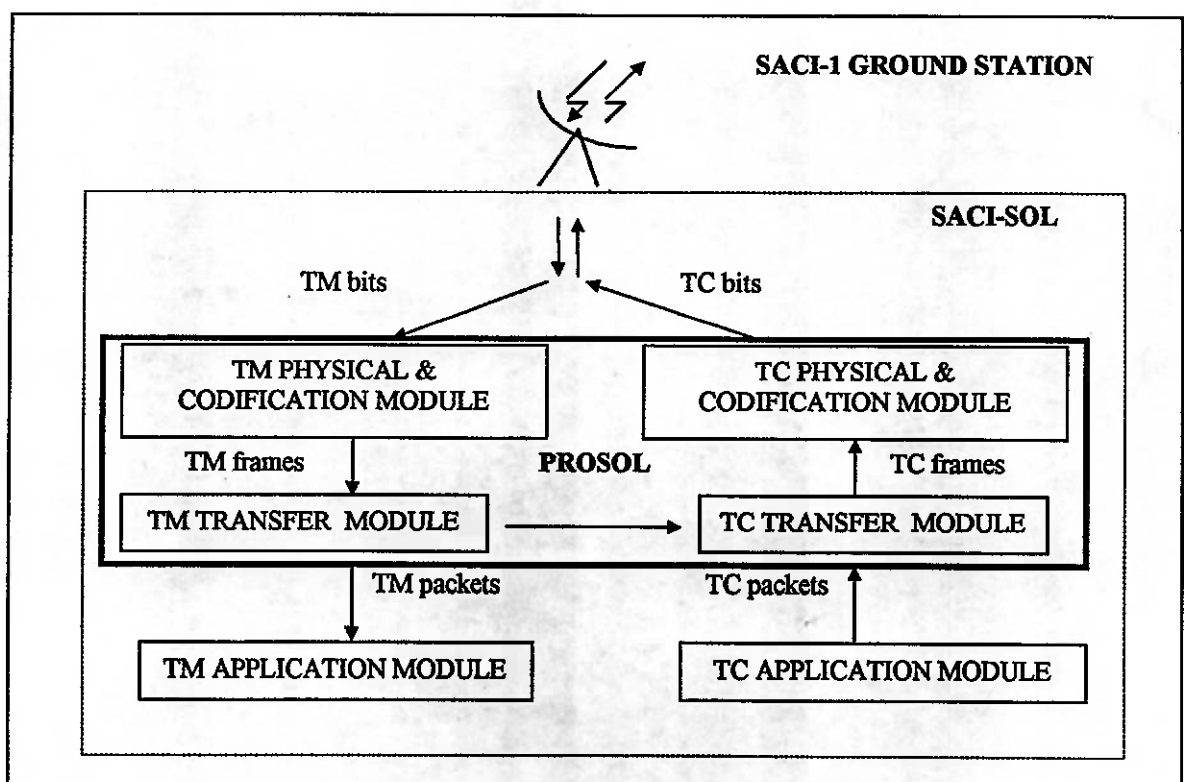


Fig. 1: SACI-SOL Modules Architecture.

2. SOFTWARE PROJECT AND DESIGN SOLUTIONS

LabVIEW development environment [NI 96] was the main project solution adopted by the developers of SACI-SOL software. Although the aim of that environment is the instrumentation, TM/TC software could be thought as remote instrumentation of a complex equipment named micro-satellite.

Since the complete architectural design of the SACI-SOL software had been pre-defined and the features of data driven graphical programming language provided by LabVIEW facilitated the prototyping, the software life cycle approach was the Incremental Delivered, as classified by ESA

in its Software Engineering Standards [ESA 91]. The software was delivered in multiple releases, each with increased functionality and capability. The bottom-up design approach combined with the graphical programming benefits of LabVIEW encouraged the program modularization by encapsulating program segments (subroutines) into reusable blocks. Many of those segments were turning in releases allowing to get both: gradual staff training in the new programming technology and performance evaluation of the critical functions.

Although, theoretically, the disadvantage of the incremental delivery approach is the high amount of testing required to confirm that existing capabilities of the software are not impaired by any new release, the potential of the virtual development tools for building quickly user interfaces allowed the isolated development and exhaustive test of each segment before being turned in block. Instead of adaptations being necessary in those blocks during the overcoming new release, what could impair them, none modification, even in the user interface, were necessary. In addition, with LabVIEW, whenever a program become a subroutine, its user interface is kept. Therefore, there is no reason for too much tests.

The project of PROSOL software subsystem followed the Consultative Committee for Space Data System (CCSDS) specification, implementing the Telemetry and Telecommand Ground-Satellite communication protocol on Ground Station up to transfer layer. Prior to any coding, it was necessary to identify the implementation component modules and the interactions among them, so that the protocol implementation could be viewed as a number of interrelated and independent processes, that communicate with each other by means of a *message passing mechanism*.

The software environment chosen to implement the protocol was also the LabVIEW. Besides its flexible graphical interface, the programming technology used in LabVIEW makes it simple. As the execution order in LabVIEW is determined by the flow of data among blocks, and not by sequential commands, it makes possible to create diagrams that have simultaneous operations. Consequently, LabVIEW allows the development of applications running multitasks, which is a necessary feature on the development of a communication protocol.

For the inter-process communication purpose, two main alternatives were placed: the DDE protocol (Dynamic Dates Exchange) and the use of FIFO queues. DDE is a proprietary protocol, developed by Microsoft, for data exchange among Windows applications in the same computer or in a local network. On behalf of the portability to other platforms and due to the simplicity in the implementation of *FIFO queues mechanisms* inside LabVIEW, the latter method was adopted for the communication between processes within the Transfer Layer and between this layer and the neighbouring ones. That is the case of the communication with the application processes, implemented by the TM and TC Application Modules as showed in figure 2. So, messages sent from a process are placed in communication queues and are asynchronously extracted by the consuming process. Each communication channel is then implemented by a pair of queues, one for each direction.

Concerning to the automatic control requirement for TC transmission, which aims to reduce the number of staffs in the Ground Station, the design solution adopted in the TC Application Module was the finite state automata. In that finite state automata, the states represent the situation in which the system stays after treating a specific event and waiting for a next one. The events are the occurrences which make the state machine change from on state to another one. And the actions are the procedures which must be executed to take the state machine to the next state.

The occurrences considered relevant for the TC automatic control were:

1. Beginning of the passage;
2. Ending of the passage;
3. Time to transmit the next TC to the micro-satellite;
4. Information related to the TC local protocol services;
5. Out of sequence TM frame;
6. Information related to the TC protocol received from TM frames.

The Automata keeps the state table data with states, events and actions. When the Automata starts to run, it is set to the initial state, initialize the monitors, queues and calls the Read Event to get event. The Read Event stays running until getting an event. The Read Event returns the event to the Automata, it consults the table in order to determine what action must be called and what is the next state. The right action is executed and, after that, both the current state is changed and the Read Event is called again by the Automata.

Still in figure 2, there are some actions that send data to the TC Transfer Layer through the queues named: Fast Request, Controlled Service Request and Management Service Request. The data contain the next TC listed in Flight Plan which is being transmitted at right time to the micro-satellite.

3. IMPLEMENTATION

This section describes some solutions adopted in the implementation of the TC Application Layer.

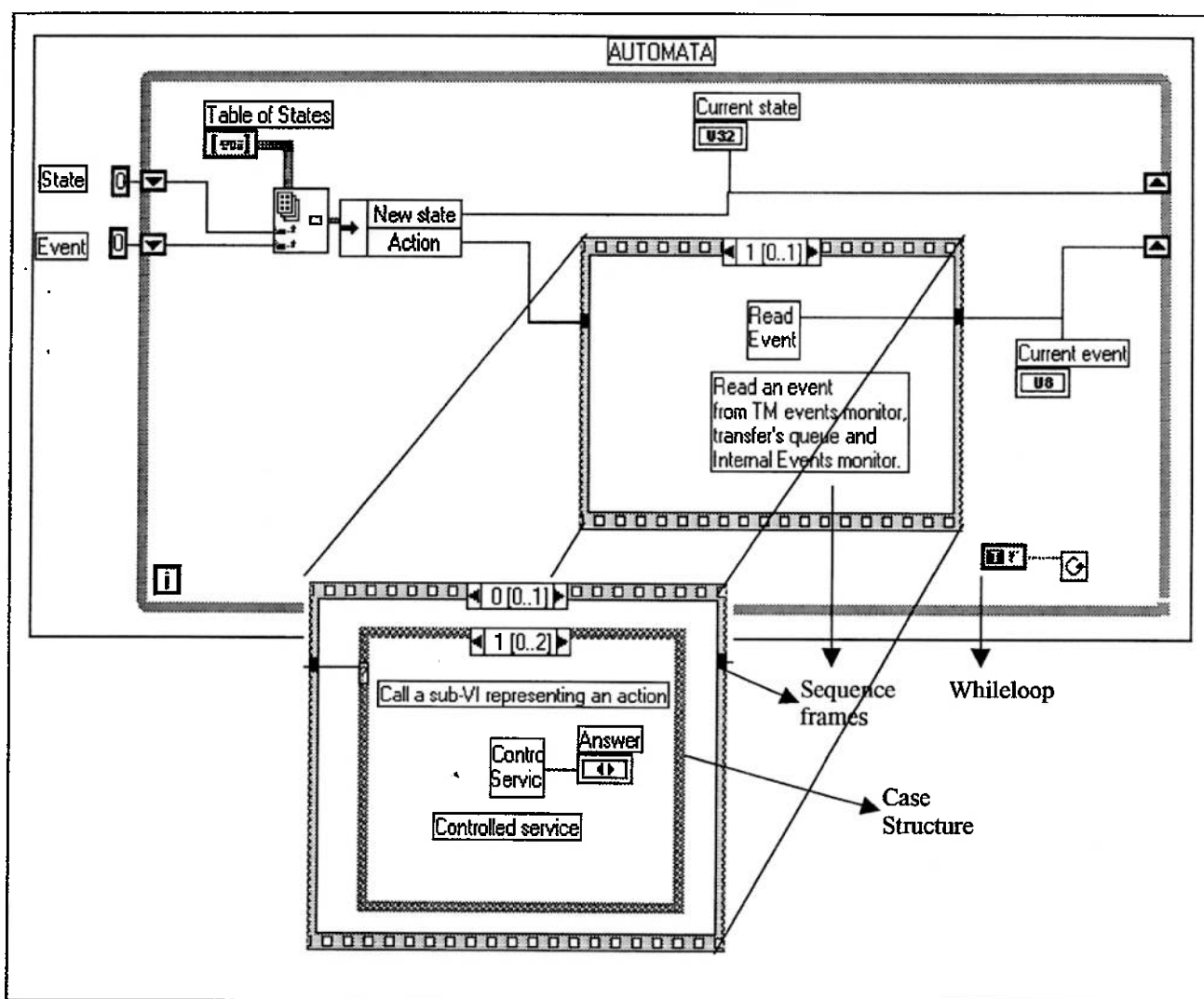


Fig. 3: Automata System in LabVIEW.

The Monitors are implemented as *concurrent processes* which are independent programs running full time. The communication among them and with other processes is done by *FIFO queues mechanisms*. So that there are 2 independent processes implementing the monitors TM Events and

Concerning to data sharing through the queues, it was used the LabVIEW default configuration which permits the memory data sharing. So, the values of the variables in a sub-routine are available to any processes calling this sub-routine. That facility was used to implement the FIFO queues mechanisms.

The queues provide the communication among several modules of the system and are based on fixed size matrix structure as circular buffer. The implementation code of the communication queues is described bellow. The insertion code is shown in figure 4.

The queue VI uses three parameters: "Opcao", "Tamanho Máximo" and "Entrada". The first one must be specified and the others are optional. The "Opcao" parameter has the following values: "Inicializa", "Insere", "Retira" and "Status".

To insert an element into the queue, the first task is to compare the current size with the maximum permitted size. If the maximum size is already been reached, no insertion is done and it returns "False". Otherwise, the element is inserted, the current size is incremented and it returns "True" in the output variable called "Sucesso".

In order to obtain an element from the queue, the first step is to verify that the queue is not empty. In case of an empty queue, the current size is zero and the output parameter is returned "False". Otherwise, the element in the queue's top is returned in the output variable called "Saída", the current size is decreased and the current top queue pointer is updated to the next element and the "True" value is returned in "Sucesso" output variable. The code for obtaining an element from a queue is shown in figure 5.

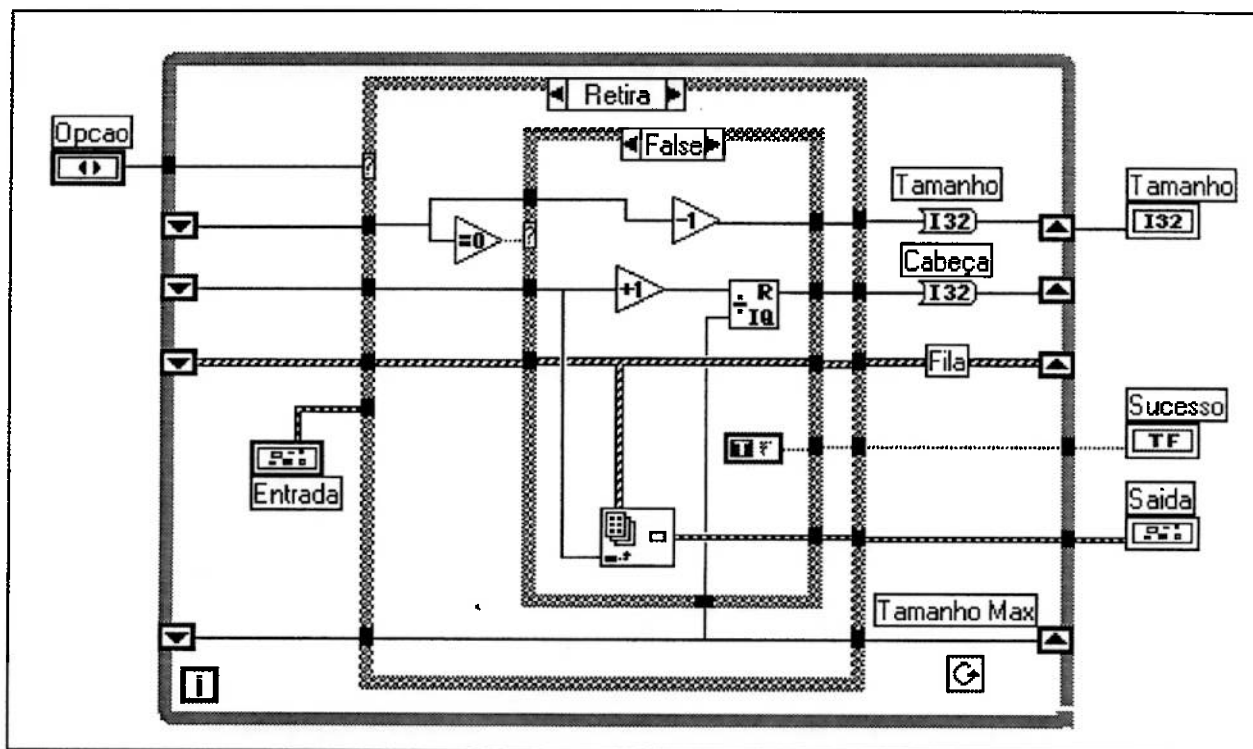


Fig. 5: Code for obtaining an element from the queue.

4. SOFTWARE TEST TOOLS FOR SACI-1 INTEGRATION PHASE

The most important consequence of using prototyping and incremental approach combined with LabVIEW programming adopted in SACI-SOL was the quick and easy development of testing

tools. Those tools allowed, during the SACI integration phase, the testing of the Ground-satellite protocol, the micro-satellite telecommand/telemetry functionality and the scientific experiment communication tasks simulating the On Board Computer functionality.

In TC context, two test tools have been done. The first one aims to test the TC reception and On Board Computer processing. Following that strategy, the transfer layer was extended with some flight plan services. The second tool intends to validate the ground strategy of unattended control of TC transmissions. Working on that, the transfer layer was integrated to the application layer in a simple way, by using a simplified automata which implements only some parts of the final TC ground system. The parts related to automatic TC transmission based on the TC transmission time stated in the flight plan. It is important to note that, the transfer layer module has been totally reused in the test tools what has contributed so much to the test of the transfer layer itself.

Moreover, a test tool was developed to validate the functionality of each onboard experiment earlier than its integration with the OBC. This test tool simulates the OBC interaction with those experiments. As a design solution, a common module for all experiment was developed to implement the OBC/Experiment communication protocol. This kernel was reused and easily adapted for each of the four experiments.

Concerning to TM software, a TM testing tool was developed to validate the reception rate requirement, an average of 60 packets/second (1 packet corresponds to 1Kbyte in size). In order to implement this requirement, the interaction of LABView programming with the TM board, the TM testing tool receives packets through "out ports" and "in ports" instructions, saving them in a file and checking if there are no lost packets. In case of a lost packet, it notifies the user.

Applying the TM testing tool, in the first test case, under Windows95 platform, LabVIEW version 4.0.1 and no user interaction, none packet was lost. Whenever the user interacts with Windows95, the tool loses packets because Windows95 is forced to attend user's request. Another test case using LabVIEW 5.0 and Windows95 was done resulting no performance improvements, even though making use of multithreading processing.

In order to overcome this problem we tried to change the platform from Windows95 to Windows NT 4.0. It does not work because the Windows NT does not permit to run common "out port" and "in port" instructions, beyond requiring more processing time for itself. Under Windows NT, the TM testing tool lost frames every time.

Therefore, we implemented part of TM software using LabWindows/CVI [CVI 96], a new development strategy for building virtual instruments with C programming language. CVI is easy to use without sacrificing any power and flexibility of the C language itself which compared to the LabVIEW graphical programming (high-level programming language) gives much more control of the code generation to the programmers than the code generated automatically by LabVIEW. That solution resulted no packet lost, even under Windows NT.

5. CONCLUSIONS

The mentioned solutions adopted on SACI-SOL system, not proved as the best one at all, allowed us to both reach the performance requirements and build testing tools quickly. Actually, the main benefit was the very short software development time (less than 2 years) despite of the small team (3 System Analysts).

Due to the facility supplied by LabVIEW for implementing user interfaces, we could reuse the prototypes as testing tools during the integration phase of the satellite.

Concerning to the modeling of TC application tasks as a finite state machine, it provided us the facility of reusing this software for other satellite control ground application. In order to reuse it, only the values in the table of states have to be changed. The automata code can be totally reused. Moreover, depending on the similarity among the new satellites and the SACI-1 satellite-ground

communication protocol, only the actions related to the application level have to be implemented. We could experience this flexibility working on testing tools.

On the other hand, on the TM context, we verified that the high rate of receiving packets requirement not matched by LabVIEW, occurred because there is no control of the amount of code generated automatically by LabVIEW. Using CVI tool we overcame that problem without leaving the LabVIEW environment, since CVI program can be called from LabVIEW as a sub-routine.

In order to improve the rate of receiving packets, a TM board driver for the home made board shall be developed. On the near future we will work on that direction.

REFERENCES

- [ESA90] ESA-PSS-04-107 Packet Telecommand Standard, *European Space Agency, Issue 1*, June 1990
- [NI 96] LabVIEW User Manual, *National Instruments Corporation Technical Publications*, 1996
- [ESA91] Software Engineering Standards, *European Space Agency*, 1991
- [Car 97] Carvalho, M. J. M. Implementação e Testes do Protocolo da Camada de Transferência do Sistema de Telecomando da Estação Solo do Satélite SACI-1. *M.Sc. Thesis*, PPgEE/UFRN. Natal, Brazil, 1997
- [CVI 96] LabWINDOWS/CVI-C for Virtual Instrumentation-User Manual, *National Instruments Corporation Technical Publications*, 1996