Journal of Computational Interdisciplinary Sciences (2011) 2(2): 111-121

Printed version ISSN 1983-8409 / Online version ISSN 2177-8833
http://epacis.net
doi: 10.6062/jcis.2011.02.02.0038

A rule-based satellite simulator for use in flight operations planning

Jun Tominaga[1], Maurício G.V. Ferreira[2] and José D.S. da Silva[3]

Manuscript received on September 27, 2010 / accepted on August 19, 2011

ABSTRACT

A rule-based satellite simulator was conceived for the verification of flight operations plans in artificial satellite control activities. Flight operations plans contain the scheduling of real-time procedures to be executed by the ground control segment, including telecommands that must be sent for the execution of mission operations by artificial satellites. Such plans, when generated by new planner software, cannot be considered entirely reliable. The safety of the missions can be ensured by evaluating the effects of scheduled telecommands on each satellite, before their actual execution. If a simulation result indicates that an unsafe internal state is reached, planning errors can be detected and corrected until an acceptable plan is obtained. The simulator consists basically on a rule-based inference engine and its associated database files. The dynamic behavior of the internal state of a satellite is defined by a set of rules. These rules are processed by the inference engine at each simulation step, in order to update the internal state parameters. The database files contain the rules that describe the system dynamics, the parameters that represent the internal state, and a queue of events, which includes the telecommands obtained from the operations plan under test and orbital events forecast by flight dynamics experts. This work describes the simulation architecture, the activities performed by the inference engine, and the data structures conceived to represent the knowledge inside the database.

Keywords: scientific computing in multidisciplinary topic, scientific computing for general applications, simulation, artificial satellites, expert systems.

1 INTRODUCTION

INPE (National Institute for Space Research) has been controlling satellites since 1993. Control operations related to satellite TT&C (Telemetry, Tracking and Commanding) are performed by the Satellite Tracking and Control Center, which comprises a control center and two TT&C ground stations. A map (Fig. 1) shows the locations of INPE Satellite Tracking and Control Center facilities. The control center is located at INPE headquarters in São José dos Campos (SJC), the main TT&C ground station in Cuiabá (CBA), and the backup station in Alcântara (ALC).

The same map (Fig. 1) also shows the orbit tracks of the satellites currently controlled by INPE. SCD1 (Data Collecting Satellite One), launched in 1993, is the first satellite designed, built, and operated by Brazil. SCD2, its successor, has been in orbit since 1998. Both satellites remain operational, after 17 and 12 years of service, respectively. These satellites relay environmental information data, acquired by a ground network of data collecting platforms from all over the Brazilian territory, to the TT&C ground stations at Cuiabá and Alcântara, from where the data is forwarded to a data center, also maintained by INPE, for data processing and distribution of products to end users.

Space-ground data exchange is performed when satellite and ground antennas are within visibility range of each other. In the map (Fig. 1), the visibility ranges of the tracking antennas in CBA and ALC are shown as circles centered at each ground

Correspondence to: Jun Tominaga – E-mail: jun@ccs.inpe.br
INPE, São José dos Campos, Brazil. E-mails: jun@ccs.inpe.br / mauricio@ccs.inpe.br / demisio@lac.inpe.br

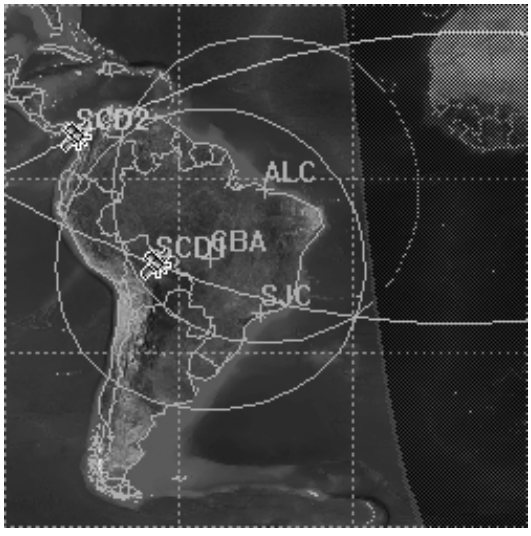

Figure 1 – INPE TT&C facilities and SCD satellites orbit tracks.

station. Real-time TT&C procedures, such as satellite telemetry data monitoring, orbital tracking measurements, and onboard status changing by telecommand, can be executed while the satellite passes over a ground station. Because such passes are defined by combining satellites orbital paths with ground stations coordinates, the execution of real-time procedures can be planned ahead in time based on satellite contact predictions. A list containing procedures scheduled to be executed from ground stations according to pass information is known as a flight operations plan.

2 SATELLITE CONTROL OPERATIONS

The control operations performed at INPE by its satellite control system are shown in Figure 2. The whole process can be considered as a sequential loop involving three control operations. Orbit measurements acquired during real-time procedures execution are used by the flight dynamics to generate satellite contact predictions. These predictions are processed by the operations planning into flight operations plans, which contain the real-time procedures scheduled to be executed during satellite passes.

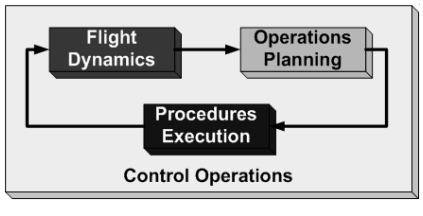

Figure 2 – The satellite control system.

Conflict management becomes necessary when a satellite passes over more than one ground station simultaneously, or when several satellites pass over a single ground station. For example, in Figure 1 the SCD1 satellite is shown inside the visibility range of both CBA and ALC, indicating that either station can track the satellite. In this case, the SCD1 pass should be assigned to one of the ground stations, since a TT&C communication link must be established between one satellite and a single ground station at any given time. In Figure 1, SCD2 is seen out of the visibility range of both. The orbits of the SCD are such that both satellites move eastward at the same rate. Therefore, by the time SCD2 moves into visibility range of ALC, SCD1 will be flying almost over SJC, inside the visibility of both stations. An easy solution to this conflict scenario is obtained by assigning the pass of SCD1 to CBA and that of SCD2 to ALC. However, such problems become more complex as the number of satellites increases. In addition to the SCD, INPE has in the past controlled satellites of the CBERS (China Brazil Earth Resources Satellites) family, provided support to India in controlling the lunar mission Chandrayaan, and currently tracks the French satellite Corot to collect and distribute its scientific payload data. Within 10 years, it is expected that INPE will be controlling more satellites of the CBERS, Amazonia, and Lattes families, as well as providing support to Chinese Shenzhou missions.

In order to address the increasing complexity of its mission operations, a new automated satellite control system is being developed at INPE. Its architecture is displayed in Figure 3.

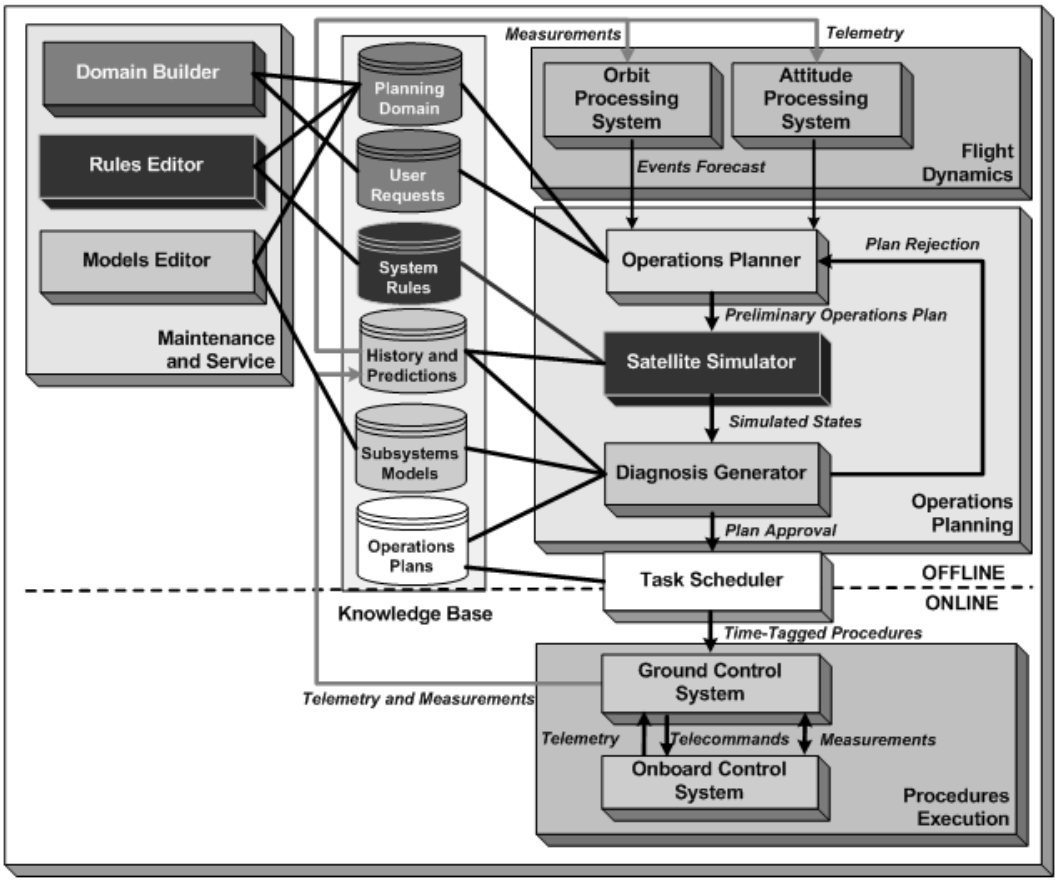

Figure 3 – The new satellite control architecture.

As shown in Figure 3, the new operations planning control operation is constituted by three parts. Firstly, an operations planner generates a preliminary flight operations plan. This plan is executed by a satellite simulator on a rule-based model, in order to test whether the plan is safe or not. This decision is made by a diagnosis generator that analyzes the simulated states supplied by the simulator, indentifies potential hazards, and then approves or rejects the plan under test. The purpose of this loop is to imbue the system with increased safety. Due to the complexity of the planning process for an increased number of missions, the operations planner may become more prone to errors. The flight operations plan must be thus validated prior to execution of its procedures in real-time.

The data flow of the newly proposed operations planning subsystem is summarized in Figure 4.

The satellite simulator is responsible for the execution of procedures scheduled in the flight operations plans under test. The operations planning data flow represented by Figure 4 evidence the interfaces required in order to run a simulation session. Input data include the flight operations plan, an orbital events forecast, and the satellite system model. These data are used to generate a simulation states history file. This states history file is then submitted for analysis, to diagnose whether the flight operations plan is safe or not.

3 THE SIMULATION ARCHITECTURE

The satellite simulation architecture is shown in Figure 5. The internal state of the simulated system is described by a set of numerical parameters. The values of these parameters at any given time determine the current state, and their behavior defines the system dynamics. The satellite simulator engine is a rule-based inference machine associated with a system rule base. This database contains the rules that describe the dynamic behavior of the satellite, as a set of causal relationships that define future values of system parameters based on their past and current values. In addition to this, a queue of time-tagged events indicates the occurrence of external triggers that affect the system. A set of values assigned to each parameter define the initial

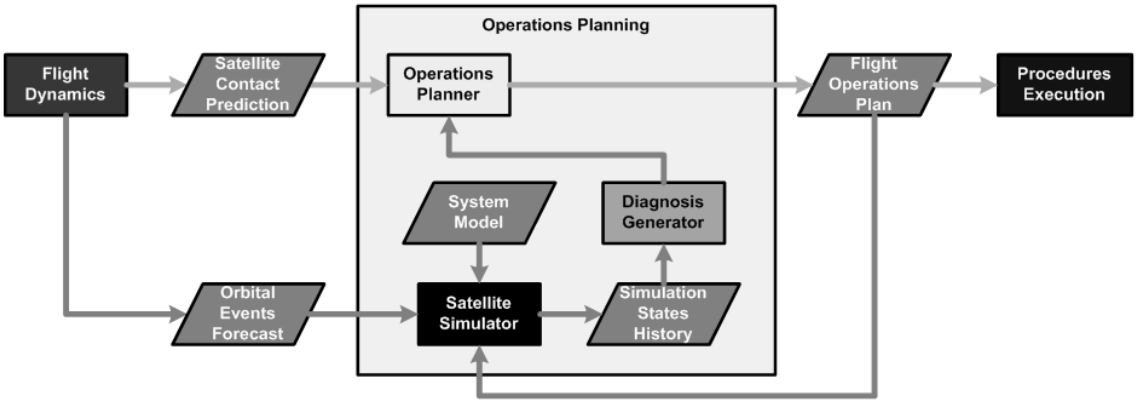

Figure 4 – The operations planning subsystem data flow.

state of the simulated system. A simulation session consists of applying the whole set of rules at regular time steps, in order to reevaluate the values of state parameters. The evolution of state parameters values, through the time interval defined by a simulation session, is recorded and exported as a simulated states history file. States history can be then analyzed to foretell how the real system should behave under the same conditions.

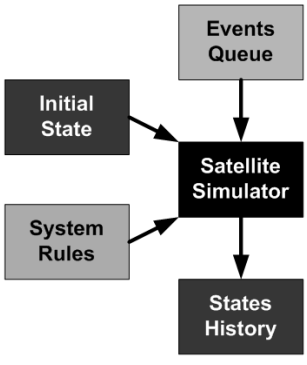

Figure 5 – Architecture adopted for the rule-based satellite simulator.

The left side of Figure 6 clarifies the relationship between the rule-based satellite simulator architecture described before and the operations planning architecture shown previously. The flight operations plan and the orbital events forecast are preprocessed into the events queue. System model rules and the initial state are created using a model configuration tool. Once this model is properly configured, the initial state can be retrieved from the simulated states history generated during a previous simulation, and system model rules can be reused in future runs. However, the events preprocessing must be executed each time a new flight operations plan is to be tested.

The simulator is ready to run after the model configuration phase is finished and the events are preprocessed. The activity diagram of the simulator is shown at the right side of Figure 6. The simulation initialization comprises five activities, during which the initial state, events queue and system rules database files are identified, imported, and converted to their respective data structures in memory. Just prior to the start of the simulation session, the simulation states history is also initialized.

Finally, the simulation session consists of three activities executed in a loop. First, the triggering of time-tagged events is checked. If the queue contains an event which execution time matches that of the simulation step, that event is triggered at that step. Such triggering is indicated by a change in value of associated event parameters. After that, all the rules are evaluated sequentially for the purpose of updating state parameters. The updated state is then written to the states history. This process is repeated cyclically, until the evaluation result of the simulation stop condition returns true.

4 SIMULATOR DATA STRUCTURES

As shown in Figure 6, during the simulation initialization, events queue, system model rules and initial state data are retrieved from database files in storage devices, and then converted to corresponding memory loaded data structures, for faster access by the simulator kernel. The database files were specifically designed to store the information necessary to build these data structures.

The system model used by the simulator is based on causal rules. They define the behavior of a satellite as understood by its design experts. Simply speaking, a rule is an IF-THEN clause, consisting of a single condition expression that must be evaluated to tell whether one or more associated effects should be applied or not. Examples of such rules are shown in Figure 7.

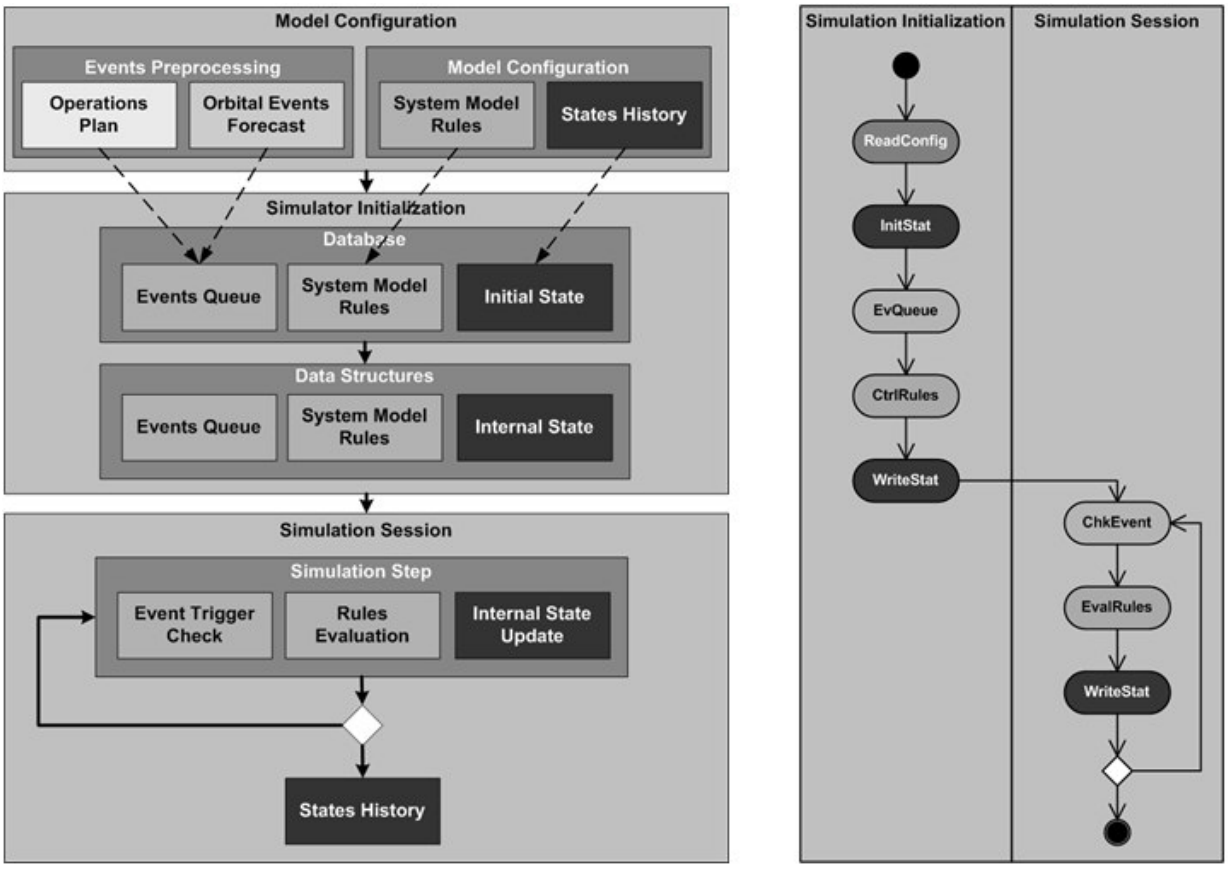

Figure 6 – The satellite simulator working phases and activities diagram.

```
IF StPower
  IF StSg02
    StWkTrxB = 0
    IF StSg01 AND StSg04
      StWkTrxA = 1
    ELSE IF StSg04
      StWkTrxA = 2
    ELSE
      StWkTrxA = 0
  ELSE
    StWkTrxA = 0
    IF StSg01 AND StSg04
      StWkTrxB = 1
    ELSE IF StSg04
      StWkTrxB = 2
    ELSE
      StWkTrxB = 0
ELSE
  StWkTrxA = 0
  StWkTrxB = 0
```

Figure 7 – Examples of satellite system model rules.

Figure 7 evidences the structures that constitute a causal rule. Basically, a rule contains a condition clause associated with one or more effect clauses. A condition clause begins with an IF statement followed by a logical expression, or an ELSE statement. Effect clauses may contain an assignment expression or a nested rule. A logical expression usually follows an IF statement in a typical condition clause, consisting of a sequence of expression elements. The evaluation of such an expression results in a logical value. This logical value indicates whether associated effect clauses should be processed or not. A condition clause started by an ELSE statement simply uses the negation of the result of the previous rule logical expression.

The evaluation process of a logical expression is exemplified by Figure 8. The upper row contains the expression, whereas the rows below contain the evaluation result at each step. The final result is shown in the bottom row.

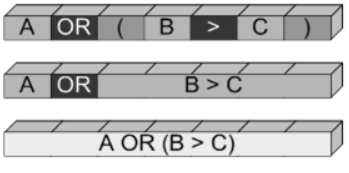

Figure 8 – Evaluation steps of a logical expression.

The logical expression shown in Figure 8 consists on an ordered set of expression elements defined by the sequence "A" → "OR" → "(" → "B" → ">" → "C" → ")". Of these elements, "A", "B" and "C" are operands, "OR" and ">" operations, and expression nesting are indicated by "(" and ")". In this example, the nested expression is evaluated first, resulting in a logical value that is used as an operand by the outer expression. Nested expressions must be evaluated in descending order of depth. In other words, results must be obtained sequentially from innermost expression towards the outermost one. This procedure also applies to the evaluation of numeric expressions associated with assignment expressions in effect clauses, as exemplified in Figure 9.

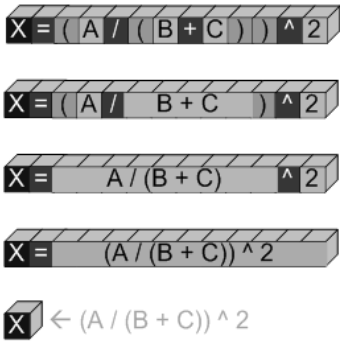

Figure 9 – Evaluation steps of an assignment expression.

Thus, an assignment expression consists of a target operand, an assignment operation, and a numeric operand. This operand is the result of the evaluation of a numeric expression that, in its simplest form, is a single numeric value. Numeric expressions are also represented as a sequence of expression elements. In the example (Fig. 9), this sequence is "(" → "A" → "/" → "(" → "B" → "+" → "C" → ")" → ")" → "^" → "2", of which "A", "B", "C", and "2" are operands, "/", "+", and "^" are operations. There are also two instances of expression nesting, shown by "(" and ")", one inside another. Since nested expressions must be evaluated in descending order of depth, the innermost expression "B" → "+" → "C" is evaluated first, followed by "A" → "/" → "(B + C)" and, at last, "(A / (B + C))" → "^" → "2". This final result is assigned to the target parameter "X".

The data structure used by the simulator to store the system rules can be thus represented as a set of expressions. Its architecture is shown in Figure 10.

Each rule comprises three structures, including an identifier, a condition, and a list of effects. A condition consists of a single logical expression. Assignment expressions that are part of effect structures are represented by a target parameter followed by a numerical expression. The assignment operation can be omitted because of the left side of an assignment expression always has the same structure. Moreover, logical and numerical expressions can be expressed using the same structure, a sequence of expression elements, as it has been previously shown. Each expression element consists of a string value that can represent an operation, an operand, or an expression nesting indicator.

Immediately prior to the evaluation of expressions, the simulator inference engine preprocesses expression elements by separating operands from operations. This expression reformatting process, in which operands and operations are assigned to different data structures, is represented in Figure 11. In the reformatted expression, the amount of operands is always made equal to the amount of operations plus one, as it will be shown.

During the expression reformatting, information about expression nesting is assigned to operations. This is done by attaching an operation depth field to each operation. This field stores a numerical value corresponding to the evaluation order of each operation. The value assigned to the depth field can be obtained by counting the number of open and closed parentheses present to the left of the operation in the expression, and subtracting one from another. This process is exemplified in Figure 12.

In Figure 12, each row corresponds to an expression, with evaluation steps indicated from left to right. The first column, the leftmost one, shows the expressions as stored inside the rules, that is, as a sequence of expression elements. Each expression element is represented by a colored block. Those with backgrounds colored in the lightest shade represent operands, darkest ones operations, and the remaining ones are opening and closing parentheses that indicate expression nesting. The numbers inside operands and operations blocks represent the index of each expression element in their respective reformatted expression data structures. The second column contains the reformatted expression, now displayed as an interleaved sequence of operands and operations. The numbers in the blocks positioned above operations are the values assigned to their respective depth fields. Operations without visible blocks are assumed to have depth value equal to zero. Expressions are evaluated sequentially from left to right, in descending order of operation depth. At each step, the leftmost operation with the highest depth value is applied to adjacent operands. This result is assigned to both operands, as indicated by the equal sign in the third column and onwards. This procedure is well suited for binary infixed operations, as shown in the second to fourth rows. For unary operations, such

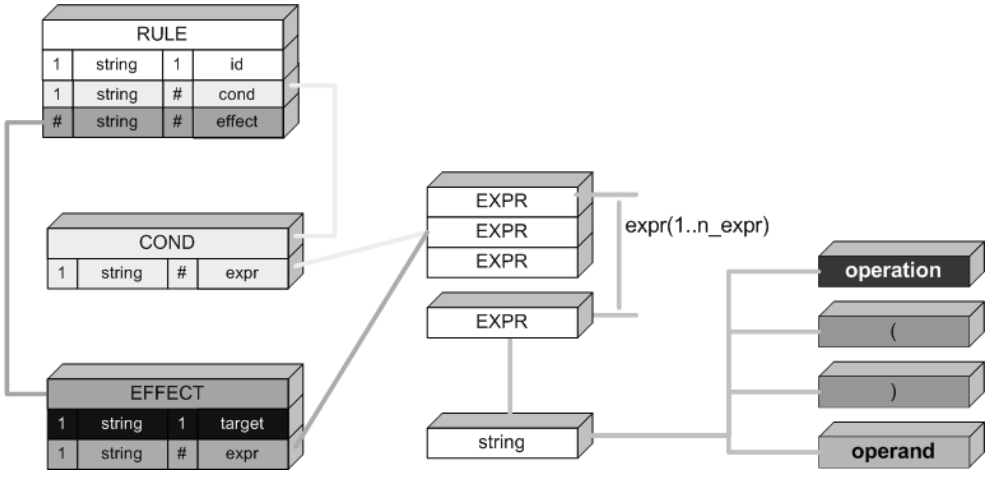

Figure 10 – Simulator rule data structure architecture.

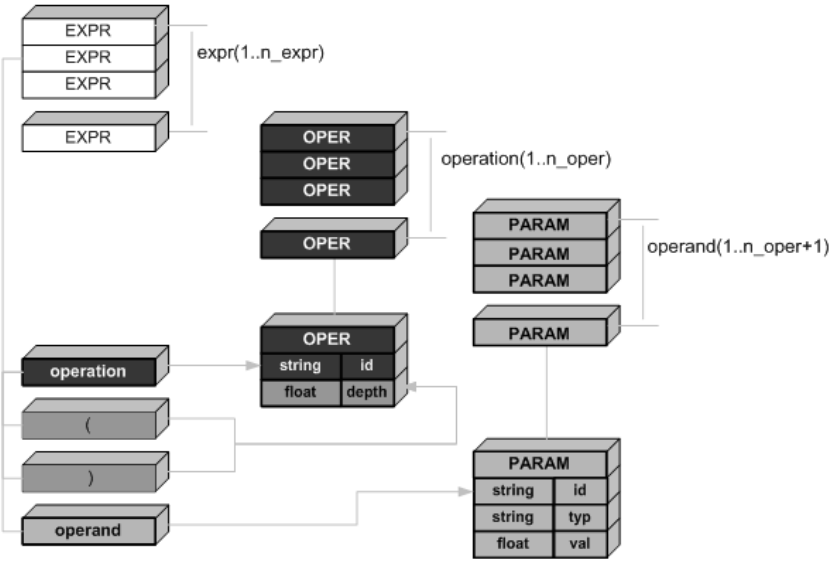

Figure 11 – Data structures of expression elements.

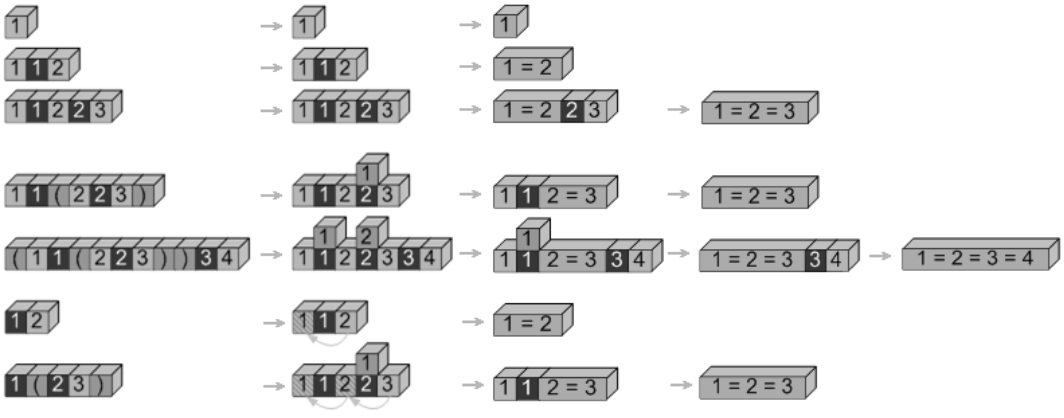

Figure 12 – Evaluation processes of some expressions.

as those found in the last two rows, dummy operands are initialized with values copied from the first valid operand, as indicated by arched arrows in the second column. At the end of the expression evaluation process, the value of the final result is assigned to all operands.

In rule expressions, an operand may represent an identifier of a variable associated with a numeric value, or a constant numeric value. In the expression "(A / (B + C)) "^" 2" (exemplified in Fig. 9), "A", "B" and "C" are identifiers, whereas "2" is a constant value. Thus, expression operands must be able to represent these two types of information. Figure 13 shows the architecture of the simulation parameter data structure, which was developed for this purpose.

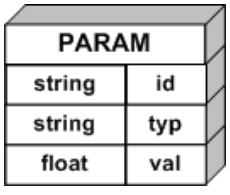

Figure 13 – Generic data structure of a simulation parameter.

A simulation parameter is constituted by an identifier field (indicated above as "id"), a type field ("typ"), and a value field ("val"). The identifier field is a label from which the content of the value field can be accessed. These two fields allow the representation of an expression operand. The type field was conceived to allow this same data structure to represent event parameters in the events queue, and state parameters inside initial state database files, as indicated in Figure 14.

As mentioned earlier, the internal state of the simulated system can be described by a set of numeric parameters, whose values define the current status. The internal state (indicated above as "stat") is, thus, represented by a set of simulation parameters of the state type. The events queue ("queue") contains a list of time-tagged triggers that, once executed, change the system behavior. The verification of events triggering is performed by comparing the simulation time, a state parameter that increments at each session step, with the execution time parameter ("exec_time") associated with each event. If these two time values match during a simulation session step, then a parameter of the event type that is associated with the triggered event is set to true for the duration of that time step. The states of parameters of the event type assume the value false while they are not triggered.

Figure 15 shows the last date structure that was conceived for this satellite simulator. It allows the representation of user functions, which can be employed to implement equipment calibration curves. Each function has an identifier used for reference inside expressions, an optional associated parameter, and a set of points. Each point is an (input, output) pair. Function output values are obtained by linear interpolation or extrapolation based on the nearest two points.

5 IMPLEMENTATION AND RESULTS

Based on the information presented so far, a simulator prototype was implemented. It was programmed in FORTRAN 77, the same language in which most of the flight dynamics software maintained at the control center was written. The modules that comprise this simulator are presented in Figure 16.

As evidenced by Figure 16, the database files were implemented using XML. System rules and their expressions are directly parsed from XML structures to arrays of strings by the program, while system parameters and function curves require additional processing for conversion to appropriate data structures. The reason behind choosing XML to design the simulator database format was based on compatibility with other pieces of software that comprise the new satellite control architecture currently under development. However, it was later found that because XML is text-based, it was well suited for manual editing of data, especially in the absence of a finished database configuration tool. A simple satellite model for testing purposes could be easily created using just a text editor.

After XML database files are converted to memory data structures, initial state parameter values are written to the output state history file for initialization. Then, at the end of each session step, the values of the internal state parameters are updated. These values are continuously appended to the output state history file, until the simulation stop condition is reached. The format adopted for this output file was another text-based one, known as CSV (Comma-Separated Values). An important factor that determined this choice was the ease of data manipulation and visualization using external datasheet tools. Figure 18 shows an example in which telemetry data obtained from a real satellite were plotted, using a datasheet tool, along with the simulated results based on a simple model of the same spacecraft. Comparisons such as this one are often performed for the purpose of mission analysis.

6 FINAL REMARKS

The rule-based satellite simulator described in this paper reflects the current state of an ongoing project. This simulator is a single component, although an important one, out of many blocks that

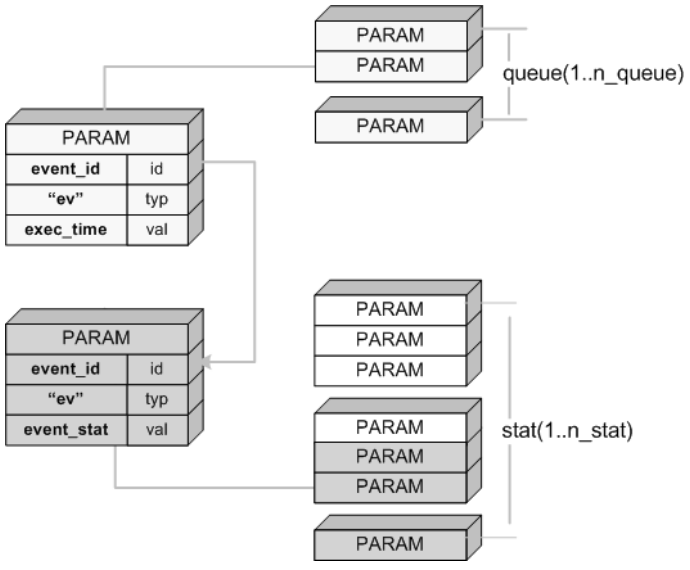

Figure 14 – Events queue and internal states data structures.

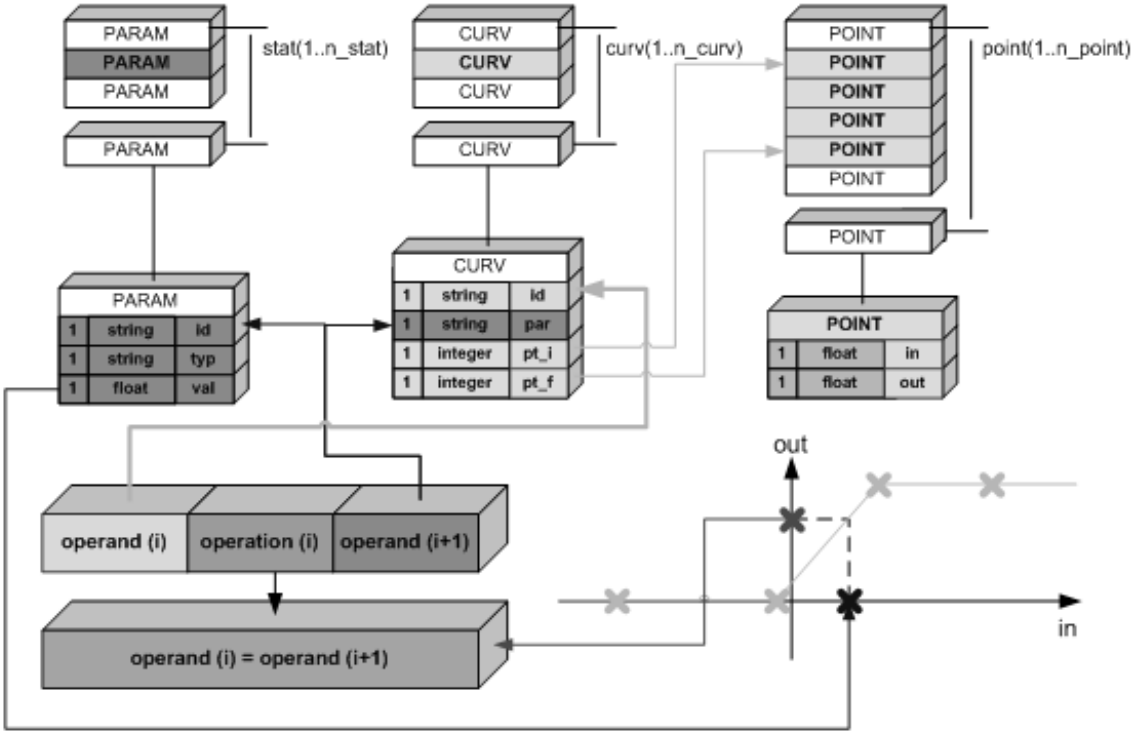

Figure 15 – User functions data structure.

form a comprehensive system aimed at satellite operations planning. The implemented prototype was proven to be fully capable of successfully simulating simple models, such as the SCD power supply model that was used to generate Figure 18. This model, which is a slightly improved version based on another one described in reference [1], was instrumental in detecting a real onboard failure, as presented in reference [2].

However, more complex models comprising hundreds to thousands of rules, required to represent high fidelity models describing a whole satellite, cannot be implemented yet. Although

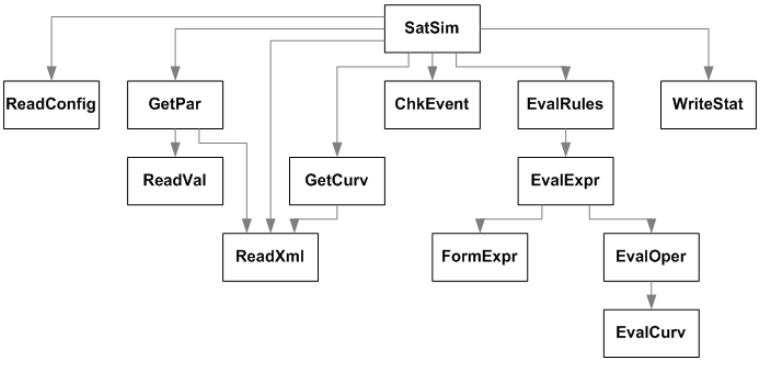

Figure 16 – Hierarchy diagram of the satellite simulator engine.

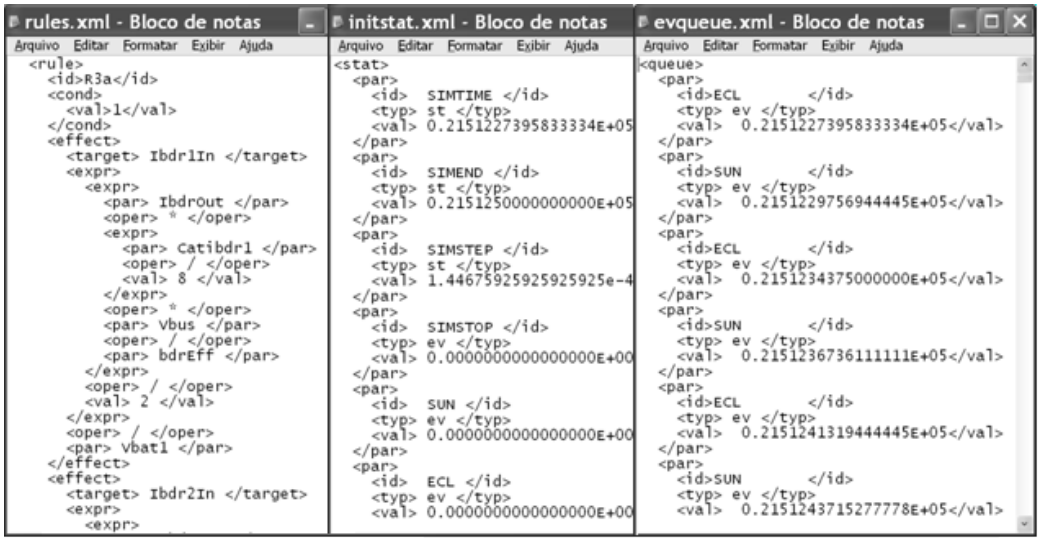

Figure 17 – Simulator database files samples.

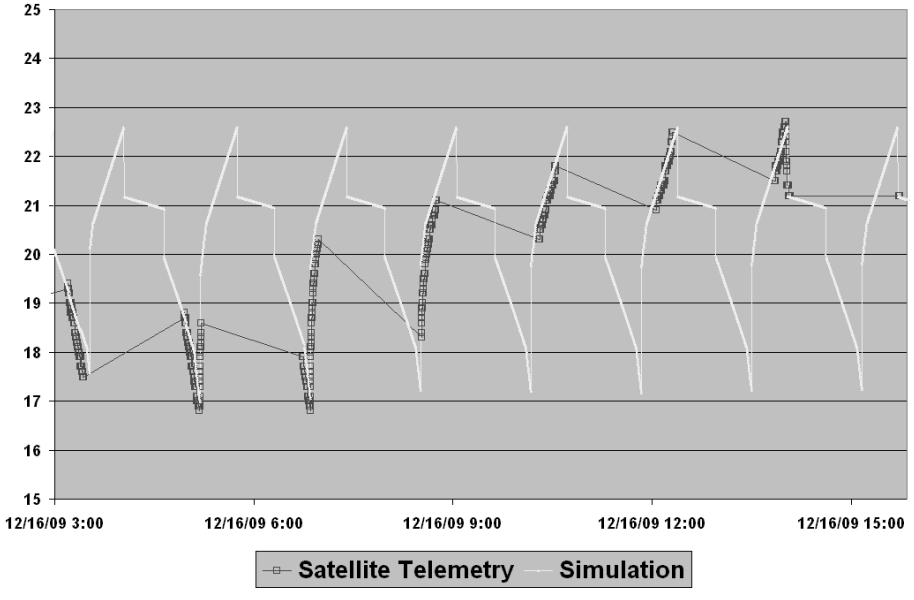

Figure 18 – Example of satellite simulation result (SCD1 battery voltage).

theoretically not impossible, implementing such intricate models correctly without the right tools for validation and testing is practically unfeasible. The correctness of the rules and the completeness of the parameters that comprise the simulated system model could and should be enforced by a rule model editor. This tool, which is not yet finished, is currently being modified to be integrated with the planning domain editor. The diagnosis generator, which shall make use of the simulator output data, is also under active development. This means that modifications to the simulator in order to implement fine adjustments to the state history file format are not yet completely ruled out.

A possible direct spinoff of this satellite simulator, developed specifically for the validation the flight operations planning, is another simulator for the monitoring of satellites at real-time procedures execution. The idea of this new simulator is to use the state history generated at the operations planning to compare predicted parameter values with satellite telemetry, in real-time. Significant discrepancies between observed and predicted values could indicate degradation of onboard equipment, thus allowing for early detection and correction of failures. The information thus gathered could be further used to refine the system model, improving its accuracy. Details about the architecture and implementation of this new simulator are currently under study. But, if approved and implemented, this new simulator could further enhance the quality of satellite control operations performed by INPE.

REFERENCES

[1] TOMINAGA J et al. 2008. A proposal for implementing automation in satellite control planning. Proceedings of the SpaceOps 2008 Conference.

[2] TOMINAGA J et al. 2010. Reconfigurable satellite simulator modeling approach for extended mission operations. Proceedings of the SpaceOps 2010 Conference, 2010.