Telecom satellites validation process evaluation

Ludivine Boche-Sauvan

LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
Airbus Defence & Space, Toulouse, France

This document originates from the memoir of Master of Ludivine Boche-Sauvan at *Institut Supérieur de l'Aéronautique et de l'Espace* (ISAE) and in collaboration with *Astrium* (now *Airbus Defence and Space*) and supervised by Bertrand Cabon at Astrium and Stéphanie Lizy-Destrez at ISAE.

Illustrations

Tables

Glossary

AIT	Assembly – Integration - Test
ANRT	Association Nationale de la Recherche et de la Technologie
CIFRE	Convention Industrielle de Formation par la Recherche
CSP	Constraint Satisfaction Problem
EADS	European Aeronautic Defence and Space Company
ISAE	Institut Supérieur de l'Aéronautique et de l'Espace
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes
SST	Satellite Sizing Tool
SW	Switch
TCN	Test Chanel Number (Path)
TVAC	Thermal VACuum

Table of Content

1. SUMMARY

Telecom satellites validation process evaluation

Student: Ludivine BOCHE-SAUVAN

Astrium supervisor: Bertrand CABON
ISAE professor: Stéphanie LIZY-DESTREZ

Abstract: Over the past decades, telecommunication demand has led to satellites with an increasing number of channels, therefore to an increasing complexity of any task associated to the payload such as design, test or validation of the performances. The internship has been hold in a team developing optimisation tools supporting payload validation process: engineering, execution and review. An analysis has been carried out on a real test sequence to understand where the next optimizations shall be enhanced. From this analysis, a mathematical model has been created as a first framework for a planning optimization. This optimization will be continued during a PhD for the next three years.

Introduction

The validation process consists in three steps:

Test Engineering: While tested under simulated environment in a thermal vacuum chamber, the payload must cover the different required unitary tests. These tests demand a routing of the RF signal along a specific path. During test engineering, the validation team defines these paths and the payload configurations (devices status, switches positions) associated. A configuration usually gathers several compatible paths.

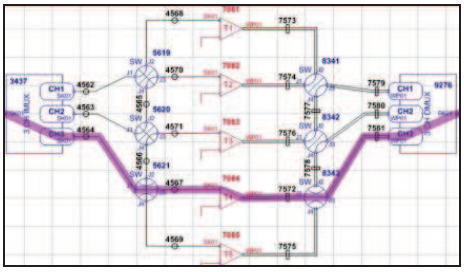


Figure 1: Path example

Test Execution: The AIT team runs the test plan delivered by the validation team. The completeness is guaranteed, however the sequenced test has to be adapted to operational constraints, such as thermal stability and other sub-systems tests. A test may even have to be redone: its configuration is then forced in the next sequences.

Test Review: The validation team checks the fulfillment of the test requirements and reports to the project management.

Reverse Engineering Analysis

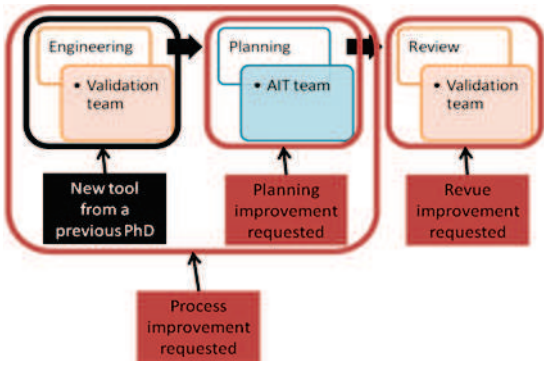
The analysis is based on real tests logs. The objective was to understand the time sharing of the AIT activities during a Thermal Vacuum phase. This analysis points out that from 10% to 30% of the payload test duration is dedicated to reconfiguration time. During this time, while switches are turning, no test can be run: this duration must be minimized. A reconfiguration time depends on the number of switches to be turned. A comparison with a sequence given by an existing tool (SST) shows the gap between reality and an ideal solution with no external constraint to follow the test plan. In this comparison, the ideal solution would need to turn switches 75% less than what is done in operation.

Mathematical model

The time spent in reconfiguration may be shortened by sequencing the paths per configuration, but also by finding the best sequence of configurations. A mathematical model has been created, including the test engineering constraints and starting from the list of paths as entering data. The variables to be optimized are the allocation of the paths in the sequenced list of configurations. This model has been integrated in a tool created during a previous PhD [1] to define paths and configuration for a specific thermal vacuum phase. A free constraint satisfaction problem solver has been used for this first approach: CHOCO [2].

Conclusion

The reverse engineering analysis gives the directions where an optimization may help reducing thermal vacuum testing time. The mathematical model is a start toward an optimization on planning. However, the optimization shall consider the different step in test process to be efficient. The next optimizations will be treated as follow:



References:

[1]C. Maillet: "Optimisation des plans de test des charges utiles des satellites de telecommunication", ISAE PhD 2012.
[2]F. Laburthe, N. Jussien : "Choco Solver Documentation", 2011

2. INTRODUCTION

Here is the report of the internship performed to end the ISAE Specialized Master of Systems Engineering. This internship was performed in Toulouse with Astrium Satellites, the European leader in space telecommunication, with collaboration with LAAS laboratory.

A lot of daily devices are today dependant to satellite telecommunication: telephones, televisions, internet... These keep evolving, demanding higher and higher telecommunication quality and capabilities on the satellites. Over the past decades, this demand has led to satellites with an increasing number of channels, therefore to an increasing complexity of any task associated to the payload such as design, test or validation of the performances. To tackle this growing complexity, tools have been developed, and maintained, to support these different tasks.

This internship aims at suggesting a new optimization tool to extend the current available optimization solutions to a more global optimization of the test problem. This is a rich topic covering different fields such as tool development, optimization algorithm and also satellite test engineering, operation and validation. The work presented here is a preparation to a coming three-year PhD with the same teams.

3. CONTEXT: PAYLOAD THERMAL VACUUM TEST

The internship's topic deals with the last phases done by Astrium on a telecom satellite before leaving to the launch site: final test under simulated environment and validation. The considered part here is the payload, element that will actually perform the mission.

3.1 TELECOM SATELLITE PAYLOAD

Here are presented what a telecom satellite and its payload are.

3.1.1 Telecom Satellite

The mission of a telecom satellite is to transfer a received signal back to Earth at a demanded power (to amplify) and frequency (to avoid any interference). To perform this mission, the telecommunication satellites need to be on the geosynchronous orbit (to be at a fixed position compared to ground stations) (3).

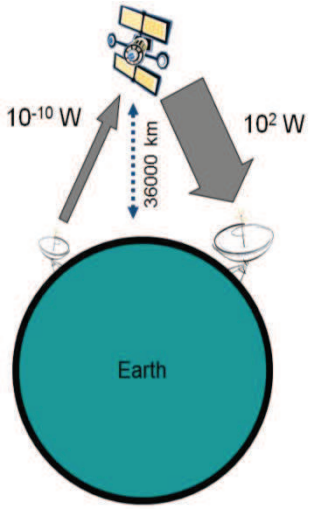


Figure 2: Satellite communication system

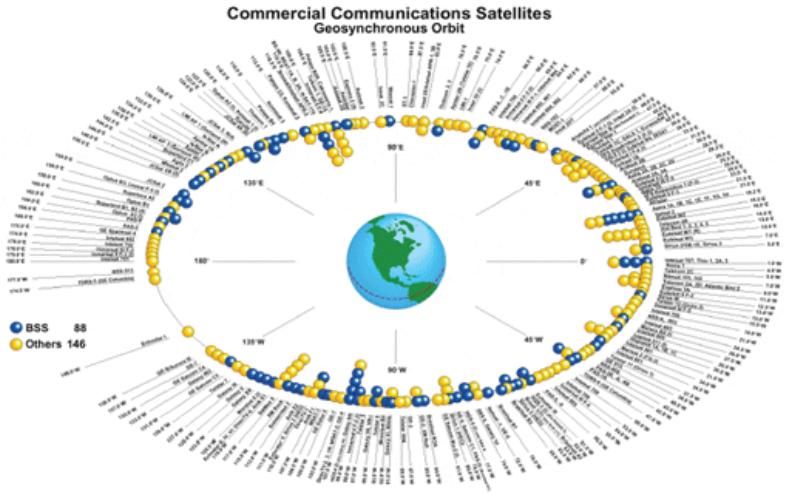


Figure 3: Geostationary Satellites by Orbital Location

The geosynchronous orbit is at 36000 km of altitude. As this is not expendable, in order to cope with the increasing demand, the telecommunication satellite's payloads are more and more complex to carry an increasing number of channels. The received signal that has spread during the travel from the Earth surface to the satellite arrives at a low energy level: around 10E-10 Watt. The satellite has to amplify it to 10e2 Watt before emitting it back to Earth, targeting the land to be covered.

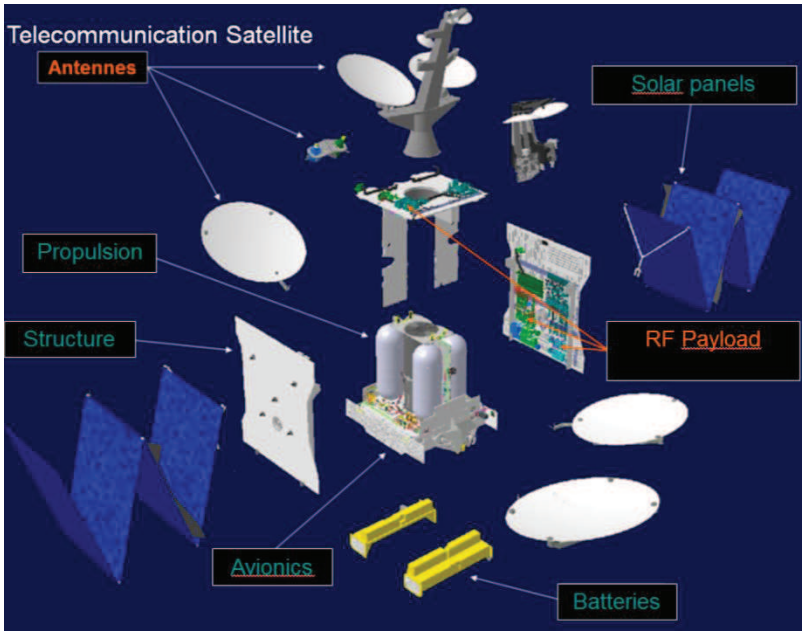


Figure 4: Satellite exploded view

The RF Payload and the antennas are the fonctional elements that will perform the mission of telecommunication. They gather the different channels and other elements (see below) that will receive, transform and emit the signal.

The rest of the satellite is the platform: it supports the payload in order to carry out the mission. The structure is the squeleton, the avionics will perform the guidance of the satellite, the batteries and solar panels provide electrical energy and the propulsion moves the satellite when it is necessary.

3.1.2 Payload equipments

The payload is a generic term considering what will actually perform the mission. In a Telecom satellite, it is the assembly of RF paths that the RF signal will cross to be re-emitted to Earth at a demanded frequency and power.

Here is the functional chain showing the different elements composing an RF path:

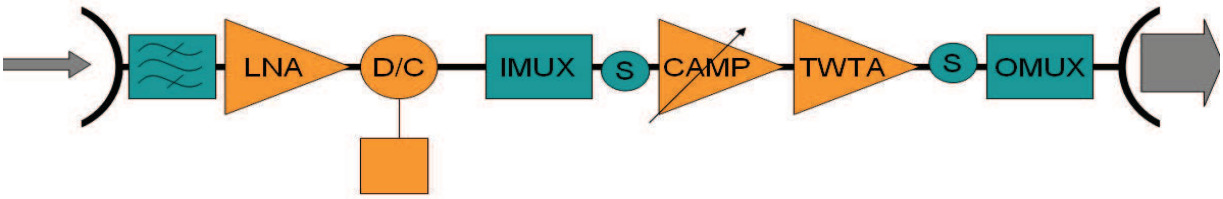


Figure 5: RF path functional chain

LNA: Low Noise Amplifier. Just near the reception antenna, it amplifies the signal and impacts it with a minimal noise.

D/C: Down Converter. Shift the received frequency into the emitting one in order to avoid any interference.

IMUX: Input Multiplexer. Separation of the different signal frequencies.

CAMP: Channel amplifier. This amplifier can be modulated from Earth.

TWTA: Travelling Wave Tube Amplifier. It is the heart of the payload. It spends more the 90% of the satellite total energy to highly amplify the signal at a narrow bandwidth.

OMUX : Output multiplexer. It recombines the signal before emission.

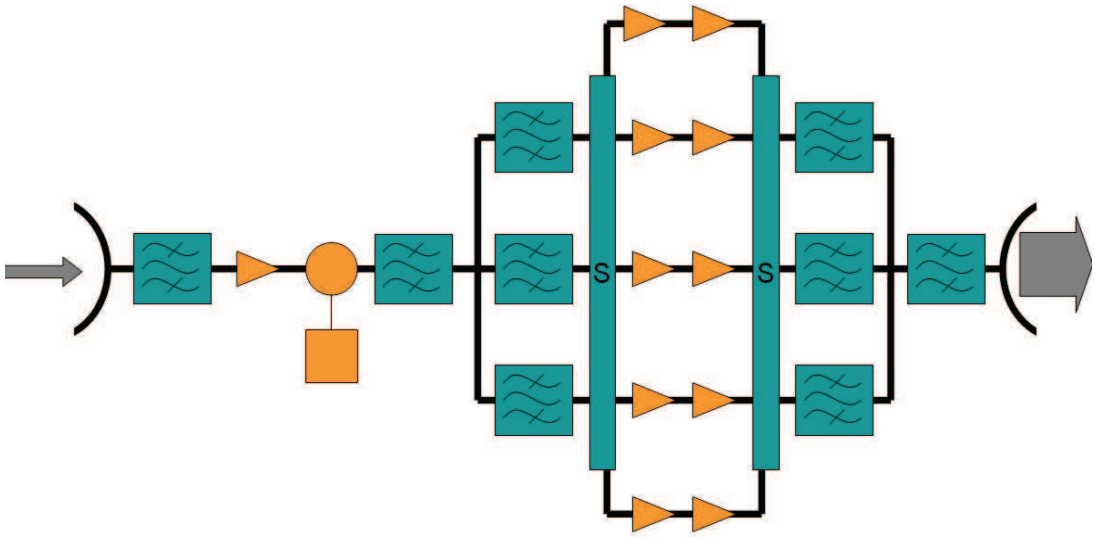


Figure 6: Redundant tubes

Added to the nominal « tubes » (CAMP + TWTA), redundant ones are added in case of failure according to the reliability required by the customer. These redundant tubes must be reachable from a certain number of nominal tubes. This implies a network of switches that will drive a signal to a tube or another. Other switch networks are also added in case of geographical selectivity: several receiving / emitting antennas to start and end a signal path. These added antennas will drive the signal to several possible targets.

Here are the different types of switch:

Tableau 1: Switch types table

Type de Switch	C		R				T		
Positions									

3.1.3 RF Path

An RF Path is the group of equipment through which a signal will go. The routing is done by the switches networks. A nominal path is a path that is used when no failure is detected. When a failure occurs on equipment, the path shall change to reach the redundant equipment without disturbing the other operating paths.

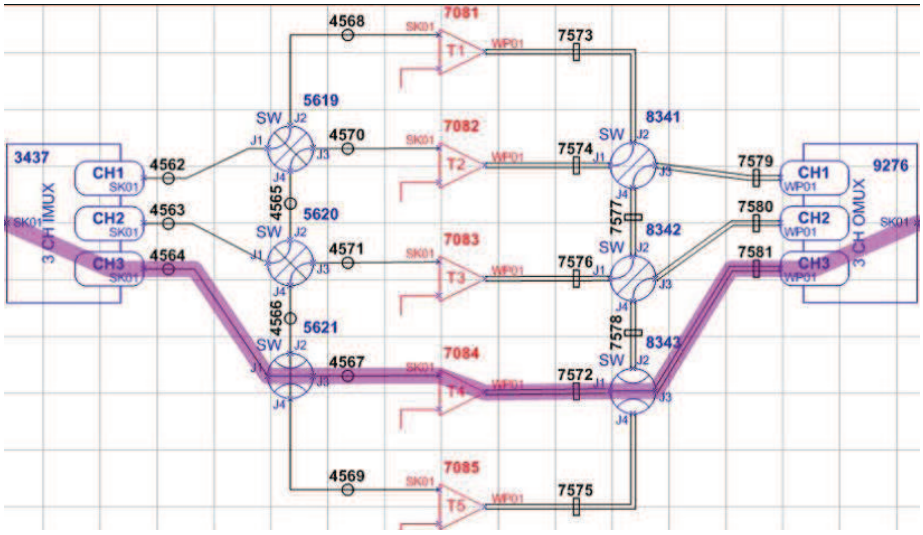


Figure 7: Path example

In tests and validation processes, a path has a number assigned and is name “TCN” (Test Chanel Number). When it needs to be ran with different gains then the TCN is extended with a decimal: the TCN (for instance TCN 138) become a subTCN (for instance subTCN 138.11).

3.1.4 Payload configuration

A payload configuration is a state of all the equipments in the payload. It defines which active equipments (LNA, D/C, CAMP, TWTA) will be ON or OFF and how the switches are turned to route the signal.

A configuration used during vacuum chamber tests shall allow several paths to be run in parallel.

3.2 THERMAL VACUUM TEST

3.2.1 Thermal vacuum facility

A thermal vacuum chamber allows testing satellite function into an environment close to space through forcing vacuum and extreme temperatures.

“SIMMER” is the biggest vacuum chamber of Intespace. With a 10 meter diameter and 8 meter length, it admits the biggest satellites to be tested. Another chamber, smaller but also used for testing full satellites, is “SIMLES”: 6.2m diameter, 7m length. Intespace has also smaller vacuum chambers, such as VTPIE (0,8m diameter, 0,8m length), allowing equipment environmental tests before integration.

These chambers simulate orbiting environment through vacuum and a temperature range from -174°C to 120°C. SIMLES even simulates sun orientation according to satellite attitude. Thermal test phases last for several weeks and may not be interrupted when vacuum restrictions are reached.

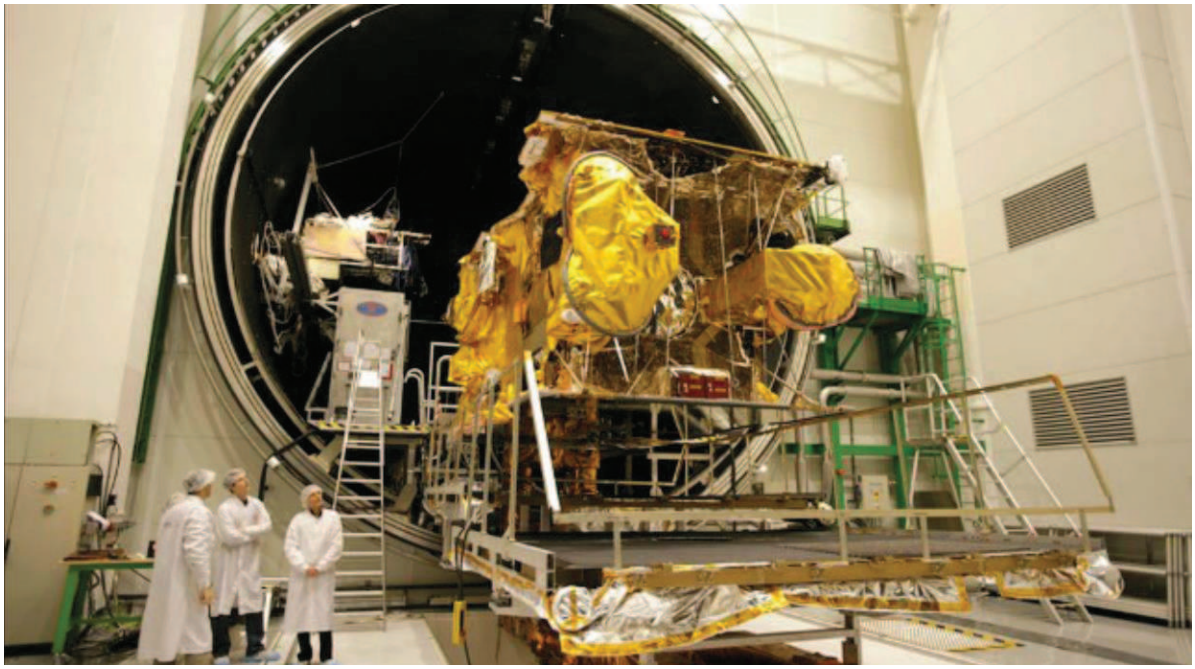


Figure 8: SIMMER vacuum chamber

The bright cover is the thermal multi layer that will protect the satellite equipments from thermal radiations impacts.

Running such a facility requires big teams under 3-shift organization and big amount of energy consumption leading to a high price/hour rate during the TVAC phase. Therefore, it is important to optimize this duration: shorten but keep the same quality level.

3.2.2 Thermal vacuum phases

Once fully built, the satellite must be tested with simulated operational environmental conditions. This is done within a vacuum chamber at Intespace (see above §5.2.1).

During the thermal vacuum test phase, the payload but also the platform are tested. The different parts of the satellite have to share a same schedule to carry out tests with global thermal constraints. This leads to "exclusive phases": payload or avionics only.

For the payload, the different tests to be conducted are provided by the Validation team according to the client's requirements. These are presented through the "Test Matrix" (see below §5.2.3) giving the different tests to be run on each path according to the being thermal phase. Here are the different thermal phases:

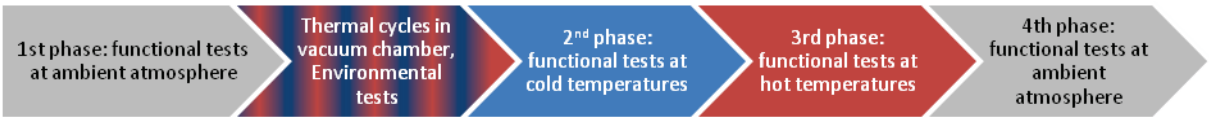


Figure 9: Payload tests phases

The strongest thermal variations happen when the satellite comes and leaves the "solar visibility" (or when it leaves and comes into the Earth shadow). To check payload equipments connectivity response to this, thermal cycles are forced between the first and second phases.

The second ("cold phase") and third ("hot phase") phases check the mission accomplishment when the extreme temperatures are reached.

3.2.3 Payload test preparation: Test Matrix

For the hot and cold phases of the thermal vacuum tests, different payload test requirements are defined by the customer. These requirements may be in terms of equipments to be run under certain tests, at a certain temperature range. From these requirements, a minimal group of paths is created in order to cover the different equipments to be tested. The test matrix will give the different tests to be applied on these paths (one line per path, one column per test).

3.2.4 Payload test preparation: Constraints

When building the paths and grouping them into configurations, some constraints have to be taken in account: compatibility, capacity, restriction and limitation constraints.

- Compatibility constraint

Any couple of paths belonging to the same configuration must be compatible: no path routing conflict (due to switch abilities).

This constraint comes with the configuration definition: it is defined by the paths' active equipments status (ON/OFF) and the path's switches positions. A switch cannot have two different positions in the same configuration: therefore, two paths allowing the same configuration must not force the switch to two different positions.

- Capacity constraint

For each configuration, the capacity constraint is the maximal number of paths that the configuration can contain.

This constraint is due to the test devices limitations.

- Restriction constraint

A restriction defines an obligation or impossibility for a path to belong to a certain configuration.

This constraint allows a finer manual definition of the configuration.

- Limitation constraint

A limitation constraint limits for any configuration the number of running active equipment (ON) in a geographical region of the payload.

This is a thermal restriction given by the thermal engineer, due to equipments thermal limitations.

3.2.5 SST product

The Engineering Product & Analysis team develops a set of products covering payload life time, from design to operations. One of them is SST: Satellite Sizing Tool.

SST is actually an Engineering product framework for telecom satellite design and validation. It contains several tools, called modules. Each of these modules has its own Excel interface. When opened, they all have access to the SST database containing all the information about all payload equipments and connexions. A VBA library had been developed, it is accessible from all the Excel interfaces, allowing for instance to get the different equipments and connexions that are in a specific payload.

Some of these Excel modules interact with the calculation Kernel, which is written under JAVA or C++. It is where the optimization algorithms are, used for example by the modules needed in payload test preparation. This Kernel uses data given by the user in the Excel interface through a text file. From this interface is also launched the solver. The user may also add in this interface constraints and test requirements. Once the solver has finished, the result is added in the Excel file (created paths, paths allocations in created configurations...).

Among the different modules, here are those related to payload validation test plans:

- Gain Monitoring: generates the TCNs that are needed for the gain monitoring test during the thermal cycles. This test checks the functional behaviour of the payload during thermal transitions. This test must cover all the different equipments under a minimal number of paths. This tool had been developed during a previous PhD (Caroline Maillet).

- TCN selection: Among a list of paths (for example those created with Gain Monitoring), this module will select the paths that will be run for tests during the COLD and HOT phases.

- Passive configuration: This module groups a list of paths into a minimal number of passive configurations. In these configurations, the paths have compatible switches positions.

- Active configuration: This module groups a list of paths into a minimal number of active configurations. In these configurations, the paths have compatible active equipments status.

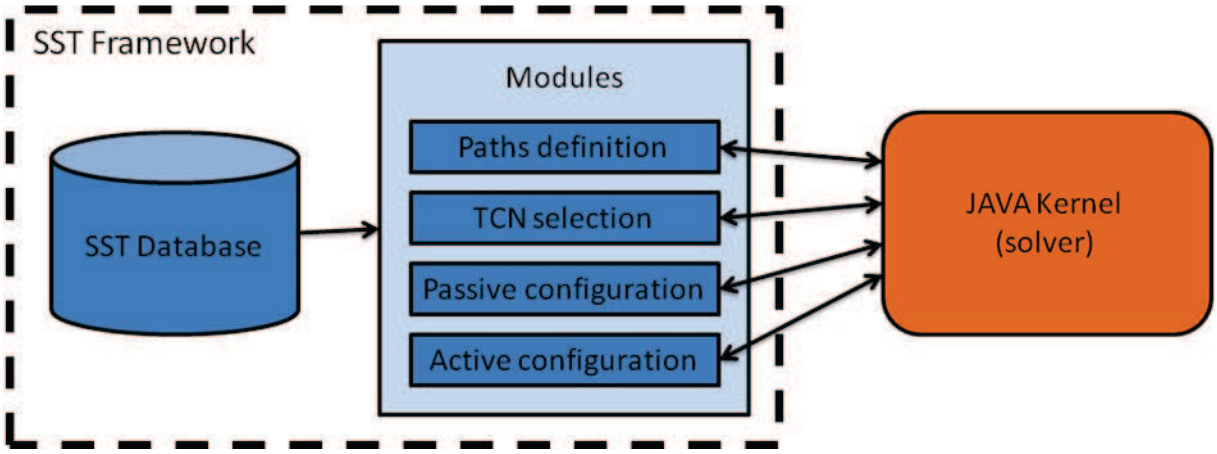


Figure 10: SST architecture

To create configurations for test plans, the process is the following:

- create paths with Paths Definition

- select paths with TCN Selection

- use Passive Configuration before Active Configuration on each passive ones if the aim is to minimize the number of used switches

- OR use Active Configuration before Passive Configuration on each active ones if the aim is to minimize the number of changing active equipments status.

At the end, the user gets a list of configurations each containing a list of (actively and passively) compatible paths. However, no test sequence is given by the tool yet.

4. OBJECTIVES

4.1 GLOBAL OBJECTIVE

The increasing demand in telecommunication quality and capacity has led to satellites with an increasing number of channels, therefore to an increasing complexity of their payloads design. This has logically complicated any task associated to the validation of their build and performances over the past decade. However, the associated tasks, such as testing, got their duration not increasing proportionally to this complexity. This has been possible thanks to a continuous development of tools and processes enabling to control and even improving project schedule without degrading the quality of the delivered product.

Telecommunication payload is a combination of many processes, continuously requiring short term optimization for different tasks to be parallelized, each of them bringing its own constraints. Moreover, each payload is unique and as such the validation process can not only be envisaged as being a predictable succession of tasks: it has to be robust to uncertainties.

Today, a review of the global process is being performed in order to define the best global strategy instead of improving only the different steps. This new strategy will surely encourage a wide use of the SST software among the different teams.

4.2 INTERNSHIP OBJECTIVES

Before defining a new process strategy, a good picture of the current one must be given. The internship has been created to this aim: to evaluate the telecom satellites validation process. The first objective is then to participate in the global one through a reverse engineering analysis and bring an overview of the current process schedule impact. A second objective is to solve the test preparation problem with an existing solver.

- Reverse Engineering

This task consisted in the analysis of TVAC and led to sub-objectives such as understanding:

- current process,
- skills, habits and expectations of the different teams,
- actual tests process.

- Solving test preparation with an existing solver

- give a mathematical model
- adapt an existing tool JAVA code to solve through the CHOCO solver

- Added objective

An underlying objective is of course to prepare the PhD with the two first ones through:

- understand the context, teams and processes organizations

- learn and code in JAVA

- understand how a solver such as CHOCO works

- learn on constraint programming and optimization

5. REVERSE ENGINEERING

Before suggesting any new process, a good inventory of fixtures must be done. New process enforcement will need time, hence the need of first improving smaller steps. This is the objective of this reverse engineering study. This study consists in analyzing logs registered during Thermal Vacuum phases by the different devices and in understanding on which steps time is spent.

5.1 DATA

5.1.1 Log

The logs to be analyzed were given by the AIT team from Toulouse after a preprocess to read them and a selection of the logs they are interested in. The given logs are under an excel format as following.

Each log has a time log and a deltatime from the previous one. It is always attached to a subTCN (or path, see §4.1.3) under a certain test. The description gives the detail of the actual log.

From this file, several data were extracted.

- Reconfiguration time

To set a TCN from the previous one, a certain time lap is spent in turning the switches (see §4.1.2 and 4.1.3) to route the RF signal through the new TCN. This is the passive reconfiguration. The active reconfiguration, or the setting active equipment under ON or OFF status, is done manual by the operator; it is considered as a hidden time.

The reconfiguration duration is the difference between the new configuration demand time and the new configuration acknowledgement time.

- Test time

The time spent on a test is clearly identified in a log description at the end of a test.

- Repetitive logs

After having extracted the reconfiguration and test times and compared them to the total time spent in the vacuum chamber, a big time period was still not yet identified. A first investigation was done in trying to identify logs that have an impact when often repeated.

5.1.2 Configurations database

A configuration is actually an assembly of local configurations of different zones of the payload. For each of these local configurations, the exact position of the switches is given in the configurations database. These local parts are called “rings” are they used to be switches circular networks.

98SW3	1298SW4	1299SW1	1299SW2	1299SW3	1299SW4	COMMENT
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 101
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 102
S 2	POS 3	POS 3	POS 3	POS 2	POS 2	IPCN 103
S 1	POS 3	POS 2	POS 2	POS 3	POS 3	IPCN 104
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 105
S 1	POS 3	POS 1	POS 1	POS 1	POS 1	IPCN 106
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 107
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 108
S 2	POS 1	POS 2	POS 2	POS 1	POS 1	IPCN 109
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 110
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 111
S 1	POS 3	POS 1	POS 1	POS 1	POS 1	IPCN 112
S 3	POS 2	POS 3	POS 3	POS 3	POS 1	IPCN 113
S 1	POS 1	POS 1	POS 1	POS 1	POS 1	IPCN 114

Figure 11: Example of a rings' configurations database

5.2 METHOD: ANALYTICAL TOOL

From the data and the configurations database, an analytical tool has been created using Excel and VBA. This tool had no aim at being developed as a widely used tool (therefore following the product development standards and validation) but as a personal tool to carry out this study. However, the AIT team is today interested in getting such a tool; this one may be kept as a first draft to be developed.

- Sequence diagram

Each TCN is defined from the configurations of the rings that it crosses. Therefore, the sequence diagram of all the rings can be rebuilt from the TCN list played as registered in the logs. Since the configuration database gives the complete status of the different rings configurations (switches positions and active equipment ON and OFF), the switches turning as well as the turning ON or OFF can be counted.

- Sequenced test matrix

For each TCN the Test Matrix defines the different tests to be conducted. With the log, it is possible to know when the tests are actually done and the status of their achievement (passed, failed, error or aborted), and the time spent on it.

The tool creates from this log a sequenced test matrix. At the cross intersection between a TCN – row (in the logs order; as they were run) and a test-column, the time spent on it is written if the test is done at this moment (if not it will be written on another line where the same TCN is run again). The colour of the cell gives the information of the test achievement status.

This part of the tool was asked by the validation team for information. From this extraction, the time sharing among the different test had been showed.

5.3 RESULTS

5.3.1 Global figures

The logs analyzed are from the COLD and HOT phases. The TCN are all run for each of the two phases with specific tests. The two phases lasted each for 11 days. However, the time gaps lasting for more than 1 hour are deliberately not taken in account as assumed with the AIT team they are a waiting time for the payload during platform exclusive tests.

Therefore, the periods taken in account for the COLD and HOT phases are respectively:

- 8 days in COLD phase

- 5 days in HOT phase

Here are pies illustrating the time sharing:

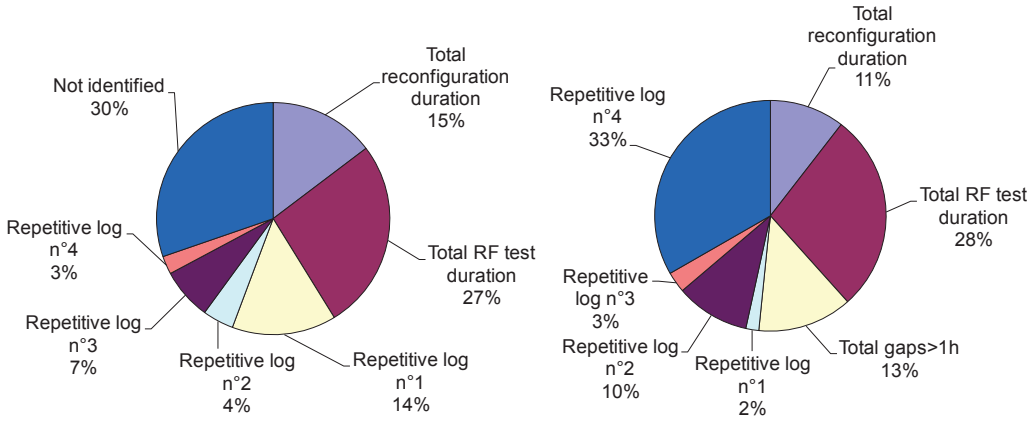


Figure 12: COLD and HOT phases global time sharing

The total reconfiguration and total RF test duration are figures directly derived from the logs (see §7.1.1) whereas the others are actually an interpretation of the rest: these are non negligible repetitive logs. A part remain not identified: they may be other logs or gaps shorter than 1h.

These figures had been welcomed during the workshop and followed by useful investigations among test devices programs.

5.3.2 Reconfiguration time

During the analysed TVAC, the active configurations (ON or OFF status of active equipments) were manually prepared in advance, so hidden in the logs. Therefore the reconfiguration considered here is only the passive one: the time the switches need to turn to reach the next configuration.

When comparing the time spent in a reconfiguration and the number of switches that need to be turned (see following graphs), it appears that the bigger amount of switches is needed, the longer the reconfiguration takes. The number of switches involved in a reconfiguration has a clear impact on the time spent.

Minimizing the number of switches involved in reconfiguration will shorten the test phases.

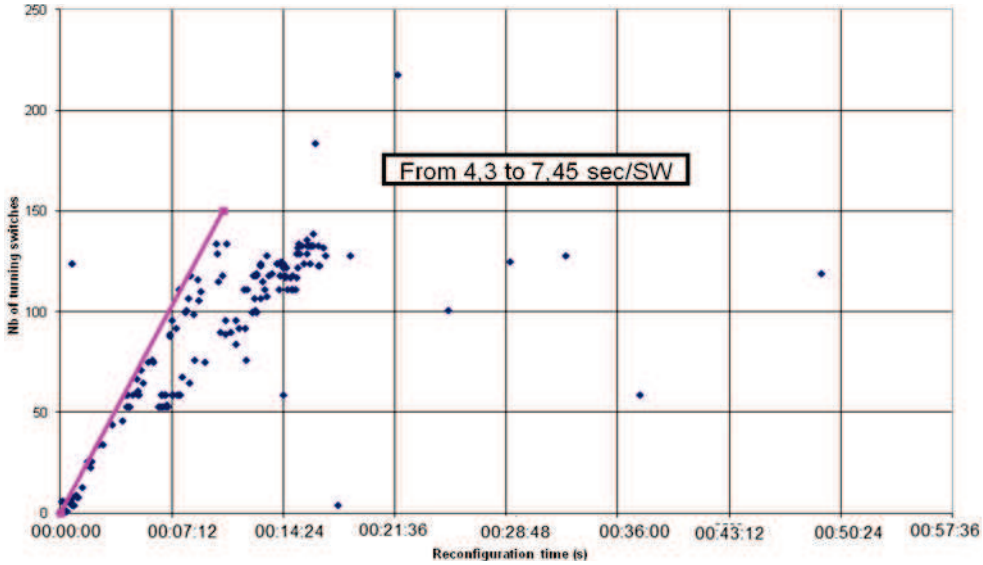


Figure 13: Switches number - Reconfiguration time Graph for COLD phase

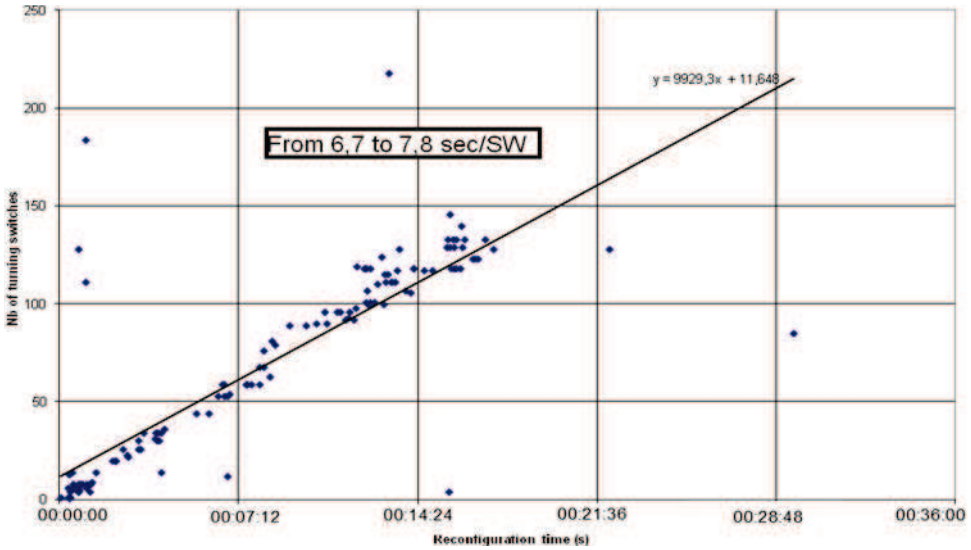


Figure 14: Switches number - Reconfiguration time Graph for HOT phase

Even if all switches do not take the same time to turn as they need to move through different increments, the average gives an idea on how a single switch may impact the reconfiguration's time.

In the COLD phase: from 4,3 to 7,45 second per switch.

In the HOT phase: from 6,7 to 7,8 second per switch.

5.3.3 Reverse Engineering Conclusion

Tableau 2: Reverse Engineering Global Figures

Phase	TVAC COLD	TVAC HOT
Phase duration	11 days 09:38:50	11 days 19:19:29
Payload activity duration	7 days 21:36:11 (~8 days)	4 days 23:07:33 (~5 days)
TCN (paths) played	377	377
subTCNs played	873	873
Nb of tests	1937	1878
Nb of reconfigurations	309 (at least passive: 180)	286 (at least passive: 149)
Number of SW modifications	14569	9726
Average time of a reconfiguration	00:05:26	00:03:53
Average number of Switches/reconfiguration	81 SW/reconf	65 SW/reconf
Average time/switch modification	From 4,3 to 7,45 sec/SW	From 6,7 to 7,8 sec/SW

Around 2 days over 22 are dedicated to passive reconfiguration. This is an important ratio demanding to be reduced. This reduction may be achieved in reducing the number of paths, or configurations, or in optimizing the sequence between the configurations.

A previous PhD (4) had already minimized the number of paths and configurations to be played. However, due to current sequence policies (nominal paths first then redundant, keeping local rings configurations building ...), and operational constraints (alive thermal constraints, possible retest required on a path), the paths are not sequenced per configurations paths groups. This leads to many reconfigurations, configurations sequence that even demand many switches moves.

- Suggested improvements:

- sequence the path list into the already optimized macro-configurations (minimize the number of reconfigurations)
  - Advantages: small number of reconfiguration
  - Drawbacks: low flexibility to operational constraints

- sequence the path list as standalone configurations (minimize the global number of switches moves)
  - Advantages: operational flexibility, low need of preparation, low number of switches moves in the reconfigurations
  - Drawbacks: high responsibility on the TVAC operator in case of retest or thermal constraints enabling the prepared sequence. The operator will have to choose among many possible configurations. A drawback also to investigate any trouble as no global big picture of the total payload configuration when a path is run (as it will depend on the total paths history).

The optimization may also find the right balance between the two suggestions:

- build more configurations with less paths

- Comparison with SST solution

The ideal would be to strictly follow the configurations given by SST. Of course it is not yet possible with the current SST version due to operational constraints. To compare, a simulation has been done taking the same paths and looking not first to local configurations, but directly to the global configuration grouping the different paths. No optimization on the configurations sequence has been done; the reconfigurations are given by the random order given by SST.

Tableau 3: Reality / SST comparison

	Real HOT TVAC	HOT TVAC SST solution
Nb of passive reconfigurations	149	12
Nb of switches moves	9726	2347
Average time / switch	From 6,7 to 7,8 sec/SW	8 sec/SW (assumed with margin)
Total reconfiguration duration	18:45:50	5:12:56 (estimated)

This SST solution follows the compatibility and usual assumed limitation constraints. However it does not include any capacity, restriction or operational constraint.

This result does not reflect reality, but it gives a direction to target in optimization.

- Toward optimization

This reverse engineering study justifies the processes improvement plan targeted for the next years.

6. PROBLEM MODEL

6.1 MODEL CONTEXT

The reverse engineering study pointed out the influence of the number of turning switches between configurations on the total TVAC time (see §7). The reconfiguration time is a time while no test can be done; it has then to be minimized.

In the model, we consider having the paths already defined (optimized with the test requirements). We want to give the best sequence diminishing the number of switches moves. For this we create passive configurations and give the best order among them. Indeed, the paths with the same passive configuration, i.e. with same switches positions shall be consecutively played.

This model is built as an SST application module: the entering data are under the text file (see §5.2.2) format given from the user interface and the solver will be in the JAVA kernel.

6.2 PROBLEM ELEMENTS

6.2.1 Data

The text file containing the entering data is organized in separated lists:

- Units: these are the different equipments of the payloads (listed in §4.1.2)

- Connexions: the connexion ports of the equipments

- Paths: each path is defined as a set of units which it goes through

- Parameters: the only parameter we use here is the number of configurations available to sort the paths. As this number is usually low (a dozen), it is not an objective to be minimized but the user will try to find the lowest. This allows keeping a high speed answer of the solver.

Methods have already been defined to get the compatibility among paths and the switch positions they require.

6.2.2 Variables

Two types of variables are defined:

- allocation of the paths in the configurations
- switch positions required in a configuration

6.2.3 Constraints

The constraints from the payload test preparation (see §4.2.4) are derived here as they appear in the algorithm.

- Restriction constraint : a unary constraint
  A unary constraint is applied to only one variable at a time. Here the unary constraint is the restriction constraint: not all the configuration may be allowed to a path. This is actually given in the parameters as the configurations that are allowed to a path.

- Compatibility constraint: a binary constraint
  A binary constraint is applied on a couple of variables.
  A path allocated in a configuration must be compatible with any other path allocated in the same configuration.

- Capacity constraint

  This constraint is here a parameter chosen high enough to not be taken in account.

- Allocation constraint:

  Any path must be allocated into one, and only one, configuration.

- Thermal constraint

  Local or global limit in number of running active equipment (that will locally heat) might be forced by thermal engineering constraint. Not considered in our model as we focus on passive configuration.

6.2.4 Objective

The different configurations, each representing a set of paths, will be tested according to the test matrix. The sequence of these configurations will induce switch rotations in between each successive couple of them.

The objective is to minimize the total number of switch rotations during the COLD and HOT phases.

6.2.5 Notation

Assembly of the paths: P

Assembly of the configurations: C

Assembly of the switches: S

6.3 MATHEMATICAL MODELS

The model is implemented under JAVA using a free CSP solver, the Choco solver: see the related codes in the annexes 2. A first model has been settled before a second model needing less variables. The model remains mainly the same but the two different models are explicitly separated when needed.

6.3.1 Data

- Paths compatibility: "C"

$$\forall (p_1,p_2) \in P^2, \qquad C_{p_1p_2} = \begin{cases} 0 \text{ if paths } p_1 \text{ and } p_2 \text{ are compatible} \\ 1 \text{ else} \end{cases}$$

- Switch position in a path: "posSW"

  This depends on the type of the switch (see table in §4.1.2). The switch position in a path is determined by the path.

$$\forall\, p \in P, \qquad \forall\, i \in S, \qquad \begin{cases} \text{if } i \text{ is } C-\text{type then } posSW_{ip} \in \{1,2\} \\ \text{if } i \text{ is } R-\text{type then } posSW_{ip} \in \{1,2,3,4\} \\ \text{if } i \text{ is } T-\text{type then } posSW_{ip} \in \{1,2,3\} \end{cases}$$

According to its type, a switch may have from 2 to 4 positions. The position "0" stands for an undetermined position: the switch is not useful in the path.

6.3.2 Variables

- Path allocation: "X", or "varConfPath" in the code.

  These variable checks for each path the allocation or not in each configuration.

First Model

These variable checks for each path the allocation or not in each configuration (1 or 0).

$$\forall\, p \in P, \qquad \forall\, k \in C, \qquad X_{pk} = \begin{cases} 1 \text{ if path } p \text{ allocated in configuration } k \\ 0 \text{ else} \end{cases}$$

Second model

Here the paths variables take the value of the allocated configuration.

$$\forall\, p \in P, \qquad \exists!\, k \in C, \qquad X_p = k$$

- Switch position in a configuration: "SW", or "varSW

These variables are created only when needed, therefore only when a path belonging to the configuration k will allow a switch i position determination. Then, the variables have the same domains as the positions of the switches according to the paths using them:

$$\forall\, p \in P, \qquad \forall ki \in C, \qquad \begin{cases} if\ i\ is\ C-type\ then\ SW_{ik} \in \{1,2\} \\ if\ i\ is\ R-type\ then\ SW_{ik} \in \{1,2,3,4\} \\ if\ i\ is\ T-type\ then\ SW_{ik} \in \{1,2,3\} \end{cases}$$

6.3.3 Constraints

- Restriction constraint:

A subassembly c of configurations C is forbidden for the allocation of the path p.

First Model

$$\forall\, p \in P, \qquad \exists\, c \subset C, \qquad \forall k \in C, \qquad \begin{cases} X_{pk} = 0\ if\ k \in c \\ X_{pk} \in \{0,1\}\ else \end{cases}$$

Second model

$$\forall\, p \in P, \qquad \exists c \subset C, \qquad if\ X_p = k, \quad then\ k \epsilon c$$

- Compatibility constraint:

First Model

$$\forall k \in C, \qquad \sum_{p_1 \in P} X_{p_1 k} \left(\sum_{p_2 \in P} X_{p_2 k} C_{p_1 p_2} \right) = 0$$

Second model

$$\forall\, (p_1, p_2) \in P^2, \qquad if\ C_{p_1 p_2} = 1\ then\ X_{p_1} \neq X_{p_2}$$

- Allocation constraint:

First Model

$$\forall p \in P, \qquad \sum_{k \in C} X_{pk} = 1$$

Second model

This constraint is respected through the path allocation variable definition: for any path, it exists a unique configuration in which it is allocated.

- Switch position constraint: a switch position in a path is the same as for this switch in the configuration of the path.

$$\forall\, i \in S, \qquad \forall\, k \in C, \qquad if\ \exists p \epsilon P\ using\ switch\ i\ and\ X_{pk} = 1, \qquad then\ \ SW_{ik} = posSW_{ip}$$

6.3.4 Objective:

The objective function OBJ must count the total number of switches turning during the TVAC.

$$OBJ = \sum_{k=1}^{CardC-1} \left(\sum_{i \in S} f(SW_{i\,k+1} - SW_{ik}) \right)$$

$$\text{With } f(x) = \begin{cases} 0 \; if \; x = 0 \\ 1 \; else \end{cases}$$

OBJ has to be minimized.

6.4 RESULT DISPLAY

A display had to be built to actually get the solution. A table is created: each of its line is a configuration with its set of paths.

6.5 VALIDATION OF THE RESULT

- To check the compatibility

Before taking in account a result, it must be validated.

First, a method (see code in Annexe) has been developed to check the compatibility among the paths in a same configuration suggested by the choco solution.

- To check the rotating switches number

The second method counts the number of switches rotations from the values allocated to the variables by Choco solver. This home-made counting checks the value of the objective function given by the solver. It also check if there is no conflict between the switches positions between the configurations and the paths belonging to them.

6.6 RESULTS WITH CHOCO SOLVER

The two models have been tested on the same list of paths as the reverse engineering analysis: 377 paths (or TCN). They have been tested in three steps: a sub list of 10, 100 then the full list of 377 paths.

Tableau 4: Choco results

	First Model			Second Model		
Paths	10	100	377	10	100	377
Nb Variables	2409	161221	Lack of memory	2219	41321	148554
Nb configurations	2			2	4	12
SW rotations	6	215		6	213	1802
Time (ms)	5375	Limit(600000)		2812	Limit(250000)	Limit(250000)
Nodes	24707	52886		10079	98743	27759
Backtracks	46887	117884		17751	185462	9526
Restarts	2	0		2	1	6

The first model imposes more variables to the solver. This leads to the impossibility to solve the problem with the full past list. In second model, the time limit is the one that could be allowed for an industrial utilization: 250000 ms = 250 s ~ 4min.

Tableau 5: Comparison reality/SST/Choco

	Real TVAC	SST solution	Choco result
Paths	377	377	377
Nb configurations	149	12	12
SW rotations	9726	2347	1802
Time/SW rotation	From 6,7 to 7,8 sec/SW	8 s/SW (assumed)	8s/SW (assumed)
Reconfiguration duration (hh:mm:ss)	18:45:50	5:12:56	4:16:00

The generic Choco solver gives better reconfiguration duration than STT. This result is promising for the next solver development that will be dedicated to this problem.

7. CONCLUSION

Improving an operational process leads at been at the crossroad among several disciplinaries. Hold in the team developing operational tools, this internship led the opportunity to also interact with tests engineering and operations team, and validation team. After having discovered the fields and learnt from these different teams, a first delivery had been required: a reverse engineering analysis. This first work results in an overview of the actual time sharing during the test running. The second phase of the internship had been dedicated to developing a model of the test engineering problem solved with a free CSP solver: Choco. This exercise is a start to prepare future works in developing home-made algorithms to optimize test engineering and planning.

The presence of both engineering and operational teams is a real advantage for Astrium in Toulouse. This helps in a continuous lessons learnt application. Often interacting with each other, test management and validation teams responsible for the payload are willing to improve their process together. This improvement will be a place for tools developments from the hosting team.

8. REFERENCES

1. **EADS.** *www.eads.com.* [Online]

2. **Astrium.** *www.astrium.eads.net.* [Online]

3. **Maral, Gérard and Bousquet, Michel.** *Satellite Communications Systems: Systems, Techniques and Technology.* [ed.] John Wiley & Sons. 4th edition. 2002.

4. **Maillet, Caroline.** *Optimisation des plans de test des charges utiles des satellites de télécommunication.* ONERA, Astrium Satellites. Toulouse : s.n., 2012. Thèse.

5. *Constraint programming for optimising satellite validation plans.* **Maillet, Caroline, Verfaillie, Gérard and Cabon, Bertrand.**

6. **Laburthe, F. and Jussien, N.** *CHOCO solver documentation.* s.l. : Free Software Foundation, 2011.

7. **Dechter, Rina.** *Constraint Processing.* s.l. : Morgan Kaufmann Publishers, 2003.

8. **Verfaillie, Gérard.** *Cours Optimisation Partie Optimisation Combinatoire, 3° année ISAE.* Toulouse : ISAE/ONERA/DCSD/CD, 2008.

9. ANNEXE : JAVA IMPLEMENTATION OF THE MODEL WITH CHOCO

The model is implemented under JAVA using the Choco solver:

```
public ChocoSolver1(Context context) {
	super(context);

	pb_ = new CPModel();


	/**
	 * Create Choco Variables
	 */
	this.createVarConfPath();
	this.createVarConfSW();

	/**
	 * Create Choco Constraints
	 */
	this.compatibilityConstraint();
	this.posSWConstraint();
	this.unaryConstraint();

	/**
	 * Create Choco Objective
	 */
	this.objective();
```

9.1.1 Variables

- Path allocation: “X”

First Model:

```
private void createVarConfPath() {

	varConfPath_ = new IntegerVariable[context_.getNumberOfPaths()][Parameters.confNumber_];
```

```
	for (int p = 0; p < context_.getNumberOfPaths(); p++) {

		String varName = "path_" + p;
		Path path = context_.getPathList().getPaths(p);

			for (int k = 0; k < Parameters.confNumber_; k++) {
			if (path.getUnaryCt(k+1)==true){
				varConfPath_[p][k]=Choco.makeIntVar(varName, 0, 1);
			}else{
				varConfPath_[p][k]=Choco.makeIntVar(varName, 0, 0);
			}
			pb_.addVariable(varConfPath_[p][k]);
		}
	}
	}
```

Second Model:

```
private void createVarConfPath() {

	ArrayList<Integer> domain=null;
	varConfPath_ = new IntegerVariable[context_.getNumberOfPaths()];

	//Unary constraints included in building the variable: a path is allowed in a certain group of configurations;
	for (int p = 0; p < context_.getNumberOfPaths(); p++) {
		domain = new ArrayList<Integer>();
		String varName = "path_" + p;
		Path path = context_.getPathList().getPaths(p);

		for (int k=1; k <= Parameters.confNumber_; k++){
			if (path.getUnaryCt(k)==true){
				domain.add(k);
			}
		}
		varConfPath_[p]= Choco.makeIntVar(varName, domain);
		pb_.addVariable(varConfPath_[p]);
	}
}
```

- Switch position in a configuration: “SW”

```
private void createVarConfSW(){

      varConfSW_ = new IntegerVariable[context_.getNumberOfNetworkObjects()][Parameters.confNumber_];

      for (int k = 0; k < Parameters.confNumber_; k++) {
            for (int i = 0; i < context_.getNumberOfNetworkObjects(); i++) {
                  NetworkObject obj = context_.getNetwork().getNetworkObject(i);
                  if (obj instanceof SwitchObject) {

                        String varName = "SW_" + i + "Conf_" + k;

                        switch (((SwitchObject)obj).getType_()) {
                        case Enum.C_SWITCH_TYPE:
                              int posC[]={1,2};
                              varConfSW_[i][k] = Choco.makeIntVar(varName, posC);
                              break;
                        case Enum.R_SWITCH_TYPE:
                              int posR[]={1,2,3,4};
                              varConfSW_[i][k] = Choco.makeIntVar(varName, posR);
                              break;
                        case Enum.T_SWITCH_TYPE:
                              int posT[]={1,2,3};
                              varConfSW_[i][k] = Choco.makeIntVar(varName, posT);
                              break;
                        }
                        pb_.addVariable(varConfSW_[i][k]);
                  }
            }
      }
}
```

9.1.2 Constraints

- Restriction constraint:

  The restriction constraint is already coded in the variables creation.

- Compatibility constraint:

First Model:

```
private void compatibilityConstraint(){

	for (int k = 0; k < Parameters.confNumber_; k++) {
		for (int p1 = 0; p1 < context_.getNumberOfPaths(); p1++) {
			for (int p2 = p1 + 1; p2 < context_.getNumberOfPaths(); p2++) {
				int compatible = (context_.getPathList().areCompatible(p1,
						p2) ? 0 : 1);
				Constraint c2 = Choco.eq(Choco.mult(varConfPath_[p1][k],
						Choco.mult(varConfPath_[p2][k], compatible)), 0);
				pb_.addConstraint(c2);
			}
		}
	}
}
```

Second Model:

```
private void compatibilityConstraint(){

	for (int p1 = 0; p1 < context_.getNumberOfPaths(); p1++) {
		for (int p2 = p1 + 1; p2 < context_.getNumberOfPaths(); p2++) {
			if(!context_.getPathList().areCompatible(p1,p2)){
				Constraint c1=Choco.neq(varConfPath_[p1],varConfPath_[p2]);
				pb_.addConstraint(c1);
			}
		}
	}
}
```

- Allocation constraint:

First Model

```
private void unaryConstraint(){

	for (int p = 0; p < context_.getNumberOfPaths(); p++) {
		Constraint c1 = Choco.eq(Choco.sum(varConfPath_[p]), 1);
		pb_.addConstraint(c1);
	}
	}
```

Second Model

This constraint is respected through the path allocation definition: for any path, it exists a unique configuration in which it is allocated.

- Switch position constraint:

```
private void posSWConstraint(){

	for (int k = 0; k < Parameters.confNumber_; k++) {
		for (int i = 0; i < context_.getNumberOfNetworkObjects(); i++) {
			NetworkObject obj = context_.getNetwork().getNetworkObject(i);
			if (obj instanceof SwitchObject) {

				//SWPathList provides path using the switch i: posSW(i) <>0
				ArrayList<Integer> pathList = context_.getPathList().getSWPathList(i);
				if (!pathList.isEmpty()) {
					for (int p = 0; p < pathList.size(); p++) {
						Constraint c2  = Choco.implies(Choco.eq(varConfPath_[pathList.get(p)][k],
1), Choco.eq(varConfSW_[i][k],context_.getPathList().getSWPos(i,pathList.get(p))));
						pb_.addConstraint(c2);
					}
				}
			}
		}
	}
```

9.1.3 Objective:

```
	private void objective(){

		int NbSW=0;
		for (int i = 0; i < context_.getNumberOfNetworkObjects(); i++) {
			NetworkObject obj = context_.getNetwork().getNetworkObject(i);
			if (obj instanceof SwitchObject) {
				NbSW++;
			}
		}


	//Count Y[i][k] of the number of position changes for a switch i from configuration k to configuration k+1
	IntegerExpressionVariable Y[][] = new IntegerExpressionVariable[NbSW][Parameters.confNumber_ - 1];

	for (int k = 0; k < Parameters.confNumber_ - 1; k++) {
		int j=0;
		for (int i = 0; i < context_.getNumberOfNetworkObjects(); i++) {
			NetworkObject obj = context_.getNetwork().getNetworkObject(i);
			if (obj instanceof SwitchObject) {

				String varName = "Y_" + j + "Confs(" + (k+1) + "," + (k + 2)+")";
				Y[j][k] = Choco.makeIntVar(varName, 0, 1);
				pb_.addVariable(Y[j][k]);

				Constraint c6=Choco.ifThenElse(Choco.eq(varConfSW_[i][k + 1],
varConfSW_[i][k]),Choco.eq(Y[j][k], 0), Choco.eq(Y[j][k], 1));
				pb_.addConstraint(c6);

				j++;
			}
		}
	}
```

```
        //Creation of the objective function sum(sumY)
        //=============================================================================
        OBJ_ = Choco.makeIntVar("OBJECTIVE", 0, Choco.MAX_UPPER_BOUND);
        //=============================================================================
        pb_.addVariable(choco.Options.V_OBJECTIVE, OBJ_);

        IntegerExpressionVariable sum = null;
        sum = Choco.sum(toOneDimArray(Y));
        Constraint c5 = Choco.eq(OBJ_, sum);
        pb_.addConstraint(c5);
        }
```

```
//Method to transform a 2 dimensions array into one with 1 dimension

      private IntegerExpressionVariable[] toOneDimArray (IntegerExpressionVariable[][] Y){

            IntegerExpressionVariable[] result = null ;
            if (Y.length!=0){
                  result = new IntegerExpressionVariable[Y.length*Y[0].length];
                  int j=0;
                  for(int i=0;i<Y.length; i++){
                        for(int k=0;k<Y[i].length;k++){
                        result[j]=Y[i][k];
                              j++;
                        }
                  }
            }
            return result;
      }
```

9.1.4 SOLVE() method:

```
public void solve() {

System.out.println("SOLVER : " + this.getClass());
solver_ = new CPSolver();
solver_.read(pb_);
//=================================================
solver_.setTimeLimit(300000);
//=================================================
System.out.println("Decision Var : " + solver_.getIntDecisionVars().length);
ChocoLogging.toVerbose();

System.out.print("Launch Solver ...");
solver_.generateSearchStrategy();

solver_.minimize(true);
solver_.solve();
```

9.1.5 Check the solution

- Check compatibility among the paths of a same configuration:

```
private boolean checkcompatibility(CPSolver s) {
      for (int conf = 0; conf < Parameters.confNumber_; conf++) {
            for (int p1 = 0; p1 < context_.getNumberOfPaths(); p1++) {
                  if (s.getVar(varConfPath_[p1][conf]).getVal() == 1) {
                        for (int p2 = 0; p2 < context_.getNumberOfPaths(); p2++) {
                              if (s.getVar(varConfPath_[p2][conf]).getVal() == 1) {

                                    if (!context_.getPathList().areCompatible(p1, p2)) {
                                          System.err
                                                .println("\nERROR: "
                                                      + p1
                                                      + " and "
                                                      + p2
```

```
                                                                        + " not compatible in configuration n° "
                                                                        + conf);
                                                    return false;
                                                }
                                            }
                                        }
                                    }
                                }
                            }

                            return true;
                        }
```

- Check counting result with an home-made counting

```
                        private void checkRotationNumber() {

                            int NbSW=0;
                            for (int i = 0; i < context_.getNumberOfNetworkObjects(); i++) {
                                NetworkObject obj = context_.getNetwork().getNetworkObject(i);
                                if (obj instanceof SwitchObject) {
                                    NbSW++;
                                }
                            }

                            //ArrayList[] TableKP = new ArrayList[Parameters.confNumber_];
                            tableSWconf_ = new int[NbSW][Parameters.confNumber_];

                            int j=0;
                            for (int i = 0; i < context_.getNumberOfNetworkObjects(); i++) {
                                NetworkObject obj = context_.getNetwork().getNetworkObject(i);
                                if (obj instanceof SwitchObject) {

                                    for (int k=0; k<Parameters.confNumber_;k++){
                                        tableSWconf_[j][k]=solver_.getVar(varConfSW_[i][k]).getVal();

                                        for (int p=0; p<context_.getNumberOfPaths();p++){
                                            if (solver_.getVar(varConfPath_[p][k]).getVal()==1){
//                                                TableKP[k].add(p);
```

```
                                    if(context_.getPathList().isSwBelongToPath(i,p)){

                                          if (tableSWconf_[j][k]!=context_.getPathList().getSWPos(i, p)){
                                                System.out.println("\nERROR: sw°"+j+"  switch position conflict
in configuration°"+(k+1));

                                          }
                                    }
                              }
                        }
                        }
                        j++;
                  }
            }

            int NbRotation=0;
            for (int i=0;i<NbSW;i++){
                  for (int k=0; k<Parameters.confNumber_-1;k++){
                        if (tableSWconf_[i][k+1]!=tableSWconf_[i][k]){
                              NbRotation++;
                        }
                  }
            }
            System.out.println("Check switches rotations number: "+NbRotation);
      }
}
```