# Avionic Software Development with TOPCASED SAM

Pierre Gaufillet, Sébastien Gabel

# Avionic Software Development with TOPCASED SAM

A. Pierre GAUFILLET[1], B. Sébastien GABEL [2]

1: Airbus Operations S.A.S, M8621, 316 route de Bayonne, 31060 TOULOUSE Cedex 9

2: CS Systèmes d'Information, Rue Brindejonc des Moulinais, BP 15872, 31506 Toulouse Cedex 05, FRANCE

**Abstract**: SAM, a graphical language dedicated to functional split up activities, has been introduced in several pilot projects at Airbus to support software specification. This article presents the needs that led to its development, the main components of its toolset and the conclusions after a few months of usage in an industrial context.

**Keywords:** models, SAM, Topcased, avionics, requirements, verification.

## 1. Airbus avionics development cycle

The development process of avionics equipment at Airbus follows a classical V cycle (see Figure 1). System engineers produce functional requirements documents that are analysed by avionic equipment teams. These requirements are then allocated to hardware or software subsets.

Let's focus on the phase of software specification in this process: the first operation of the software specifiers consists in defining hierarchically a functional architecture organizing the answer to the upstream requirements. Beyond this organization, the data and control flows between the functions need to be identified.

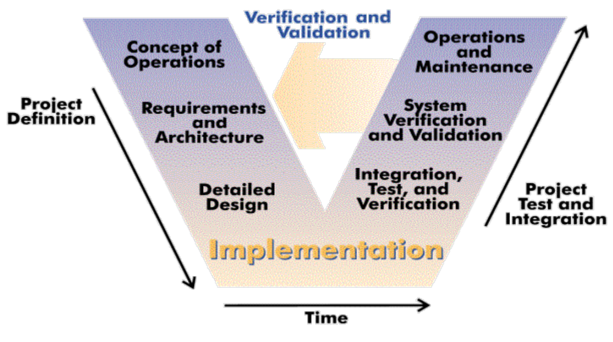

Verification and Validation
Concept of Operations
Operations and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Figure 1 - V model

Once this stage has been reasonably completed, the terminal functions of this functional split up are allocated to industrial subsets – these subsets are defined depending on various industrial constraints like teamwork, industrial partnerships, physical partition, re-use of existing components, segregation of different level of criticality, etc. Each subset is then exported to a specific team who will refine the functional split up until the required level of details. This refinement will be the basis of the subsequent technical design.

## 2. Software specification

To perform these specification stages, only a few abstractions are necessary:

### 2.1 Requirements

To ensure that the applications fulfil the needs of the customer, development choices have to be tracked step by step. Of course, those choices are more or less detailed depending on the considered stage of development, but they all cover one or more upstream needs, and will be covered by one or more lower level choices until the implementation stage. These customer needs and development choices are called requirements. They may be described informally – natural language, free-hand schemas – or formally – abstract code, state machines, activities flow.

### 2.2 Functions

Fundamental building block of these specifications, functions identifies behavioral sets featuring inputs and outputs. Functions may be decomposed in smaller functions, to ensure that the complexity of the description is limited to the strict minimum for a given level of description. Terminal function's behavior is specified by requirements.

### 2.3 Flows

Flows are a side effect of the functional split up: they bridge the gap between the functions that have been defined, ensuring the communication of events and data between them. Flows require a special care, as they will lead more or less directly to define the API of the applications.

## 3. Modeling

### 3.1 SAM history

To improve maintainability and automatize as much operations as possible, we chose years ago to follow a MDE approach to support the specification stages.

It led us to derive a simple modeling language from the SART concepts called SAM – for Structured Analysis Model. SAM has been designed to cover precisely the concepts seen in section 2, and was initially supported by Sildex, a tool based on the SIGNAL language and developed by the company Geensys. As SAM gathers only a mere 50 concepts compared to hundreds in UML, it was a good candidate in Topcased[1] to develop a complete tool chain as required by our process. As most of Topcased tools, SAM ones have been themselves developed using MDE paradigm: the graphical editor for example is based on an ECORE meta-model and has been partially generated.

When Topcased 2 was published in 2008, SAM core tools were mature enough to be deployed on pilot projects. We therefore began to add typically industrial new features and to refine the existing components with the pilot project teams. This collaboration brought a lot of improvements and some brand new features like requirement management in SAM models.

### 3.2 SAM concepts

SAM defines a limited – but sufficient – number of concepts in order to facilitate learning and understanding the models produced. The current version, SAM 1.2, includes 40 concepts, and the version to come, SAM 1.3, will grow up to 48 concepts. Only SAM 1.2 will be covered here (SAM 1.3 extends the hierarchical approach to flows description).

#### 3.2.1 Hierarchical approach

SAM, accordingly to the software specification needs described in section 2, defines a hierarchical architecture of functions. The software – or sometimes its environment - being designed is modeled as the root function. It is then successively decomposed into several sub-functions until the expected degree of detail is reached.

There are 2 kinds of function:

System: it represents a function that may be decomposed into smaller functions or not, with input and output interfaces (ports Cf. §3.2.2) connected via flows.

Automaton: closely related to Mealy state machines, Automata are a convenient mean to describe formally deterministic behaviors and therefore requirements that generate an output based on their current state and input. Of course, Automata are leaves of SAM models, and can own control, data and message input ports and control output ports. Depending on the destination of an Automaton (documentation, code / test plans / formal properties generation, model checking), data and message input ports may be forbidden.

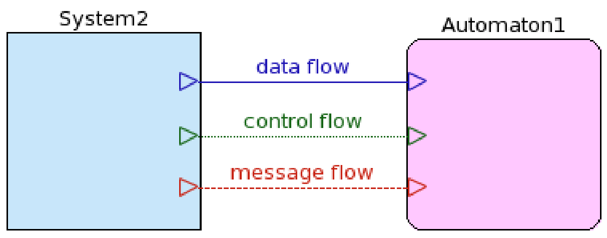


Figure 2 : System and Automaton with flows

#### 3.2.2 System children

SAM also includes the following detailed features that may be contained by a System:

Port: define input and output interfaces of the different components. Ports can be typed as data (a typed value), control (an event) or message (both typed value and event) to propagate a piece of information.

Flow: connect two ports of the same type and define communication means and synchronisation methods for systems and automata. This notion is oriented.

DataStorage: this component, once connected on a data flow, means that this flow is persistent: i.e. data are kept from one system's life phase to another.

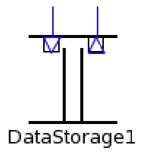


Figure 3 : DataStorage figure

Broadcast, Merge and Split: these components allow to multiply the number of flow producers or consumers.

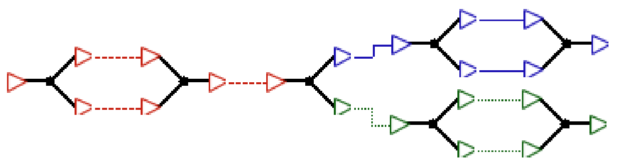


Figure 4 : Broadcast, Merge and Split figures

Composition and Decomposition: used to define structured data. The first one allows gathering several data flows, while the second extracts data flows from a composite flow.

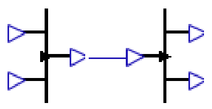


Figure 5 : Composition and Decomposition

#### 3.2.3 Automaton children

Automata are build with 2 main families of elements:

States: they represents stable situation of the Automaton. Macro states are used to factorise output transitions of several states. The initial state – only one for a given Automaton – defines as it is clear from its name the initial state of the Automaton.

Transition: each transition, connecting 2 states, has 3 attributes:

- A boolean expression including input port; this guard enunciates the firing condition. If the condition is let empty, condition is fired anyway.
- A set of output ports that define precisely the events emitted when the transition is fired.
- A priority represented by a natural value. Priorities are required to select which transition is to be fired when several are possible.

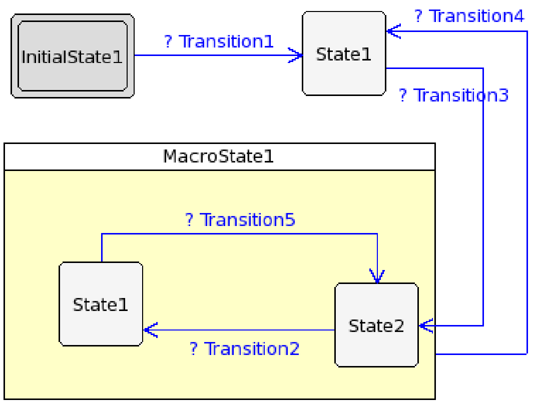


Figure 6 : Automaton diagram

## 4. Topcased SAM tools

### 4.1 Framework - EMP features

Before going further, it is essential to underline that the entire Topcased platform closely depends on most of the features provided by the Eclipse Modeling Project[2]. EMP focuses on the evolution and promotion of model based development technologies within the Eclipse community. It provides a unified set of modeling frameworks, tooling, and standards implementations.

For instance, Topcased is built around Eclipse Modeling Framework. EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. Numerous Eclipse modeling components are chained within Topcased applications: MDT OCL for static semantics checking, MDT UML for editing UML models, M2M (ATL) for defining import/export facilities, M2T (Acceleo, Jet, XPand) for generating documentation, etc.

### 4.2 Graphical Editor

Partially generated few years ago thanks to the Topcased editor generator, this graphical editor is nowadays maintained manually when modifications are required or when the SAM meta-model evolves.

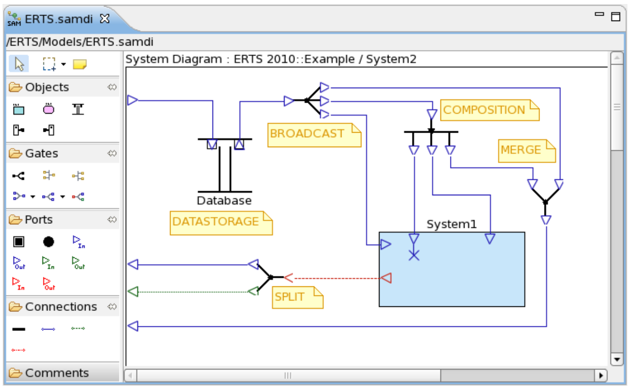
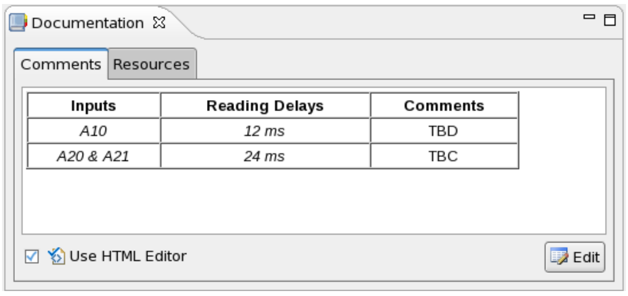


Figure 7 : System diagram in SAM editor

As in the other Topcased graphical editors, we find the following layout: a tools palette is located on the left of the editor area – its content is updated in accordance with the kind of diagram displayed. The user may arrange views around the main edit area according to its needs and work habits.

The Eclipse menu bar gives access to basic graphical features such as show/hide grid, elements alignment, snap to geometry, zoom, etc. but also t o navigation (go to next, previous, parent diagram) and model verification (EMF and OCL validations).

The SAM properties or the graphical properties of any model elements can be edited thanks to traditional views offered by Eclipse (as the Outline or Property views) or specifically by Topcased. The Documentation view for example allows manipulating complex HTML bodies such as comments or requirement description. It also enables to associate URL links or external documents.



| Inputs | Reading Delays | Comments |
|---|---|---|
| *A10* | *12 ms* | TBD |
| *A20 & A21* | *24 ms* | TBC |

Figure 8 : TOPCASED Docmentation View

### 4.3 Import Sildex legacy models

Of course, almost no tool is introduced into industrial process without taking care of legacy data. SAM tools are not an exception in that domain: porting the existing models from Sildex to SAM was a real concern.

For this purpose, a *model-to-model* transformation using ATL has been developed and integrated into TOPCASED. And as history is prone to repetition, a complete process has been implemented to ensure the migration of SAM models from each meta-model version (1.1, 1.1.1, 1.2 and 1.3) to its successor. That is why today, we are still able to import a Sildex model into a SAM model conform to the latest SAM definition.

## 4.4 Requirements manager

In 2008, a new tool aiming to manage textual requirements in SAM environment has been introduced. The goals were to establish links between textual requirements and model elements, and to describe how the current requirements are covering their upstream requirements.

### 4.4.1 Requirements concepts

Our approach is based on *TRAMway*. This component, developed in TOPCASED by Geensys, defines a meta-model that includes all the requirements concepts that may be useful in a modelling environment. *TRAMway* also provides a simple traceability solution working on a documents tree (specification, design, validation plan, test plan, etc.). An Open Office document parser is provided with this tool to import requirements, but they may also be directly injected in a XMI format.

But the *TRAMway* meta-model was not rich enough to fully describe the requirements used in avionics software specification documents. It has to be extended with new concepts such as requirement allocation; untraced, problematic and unaffected relationships requirements; and even requirement project configuration. With these extensions, a current requirement references two different elements: one SAM element and one upstream requirement.

*TRAMway* requirements importers are not yet powerful enough to process complex documentation formats. But hopefully, *Reqtify* is able to do it, and directly generate *TRAMway* models.

### 4.4.2 Requirements workflow

So requirements are managed in 3 distinct steps:

- Generate a requirement model from a document (odt, ods, etc.) or from a Reqtify export,
- Attach the requirement file to a SAM model and creates requirement links between current requirements and graphical model elements, between current requirements and upstream requirements,
- Export specification documents including schemas and requirements.

### 4.4.3 GUI

The requirement manager proposes several tools:

- The upstream view: shows the upstream part imported in read-only. This tree hierarchy is structured in documents containing sections where upstream requirements are stored. A simple drag and drop operation, from one or a group of upstream requirements to a graphical SAM element, will create one or several current requirements. Upstream requirements appear with a bold italic font style as soon as they are covered at least once by a current requirement.

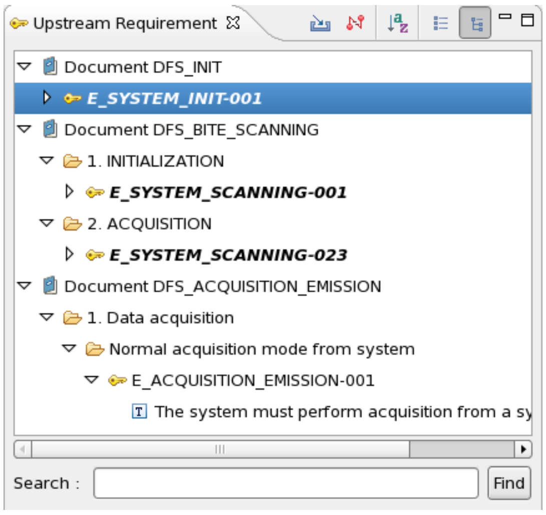

Upstream Requirement
Document DFS_INIT
***E_SYSTEM_INIT-001***
Document DFS_BITE_SCANNING
1. INITIALIZATION
***E_SYSTEM_SCANNING-001***
2. ACQUISITION
***E_SYSTEM_SCANNING-023***
Document DFS_ACQUISITION_EMISSION
1. Data acquisition
Normal acquisition mode from system
E_ACQUISITION_EMISSION-001
The system must perform acquisition from a sy
Search : Find

Figure 9 : Upstream Requirement view

- The current view: presents a set of current requirements and free texts in a tree reflecting the SAM model itself. Requirements are organized accordingly to their SAM model container. For each container, it is possible to alter the order of requirements and free texts. The document generator will later respects this order in each container section.

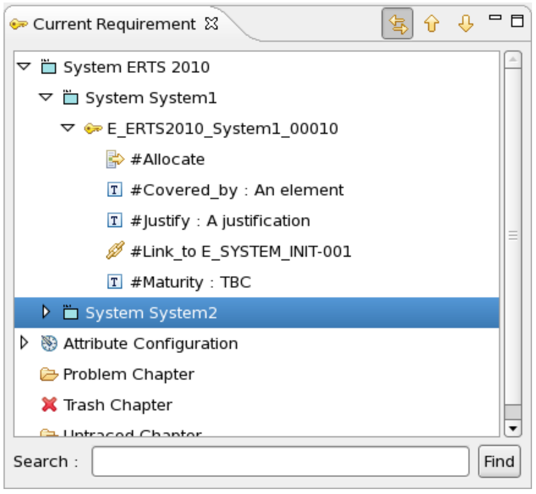
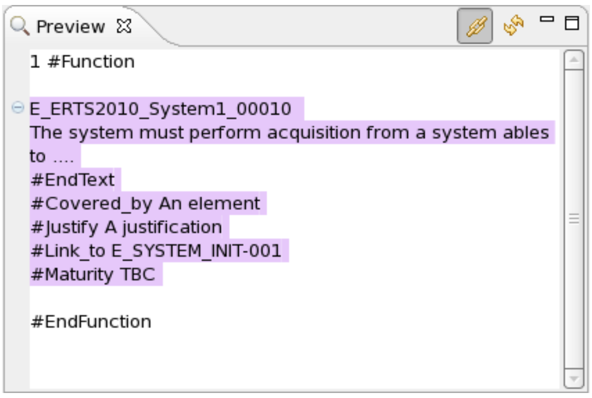

Current Requirement
System ERTS 2010
System System1
E_ERTS2010_System1_00010
#Allocate
#Covered_by : An element
#Justify : A justification
#Link_to E_SYSTEM_INIT-001
#Maturity : TBC
System System2
Attribute Configuration
Problem Chapter
Trash Chapter
Untraced Chapter
Search : Find

Figure 10 : Current Requirement view

- Preview: provides a real document generator limited to raw text. This view allows the user to get on the fly a projection of what will be the final document. The text of this view stays connected to its predecessors and its content is linked to the current selection. A double click on a line or a text block leads to the requirement model element (synchronized with current view).


Preview

```
1 #Function

E_ERTS2010_System1_00010
The system must perform acquisition from a system ables
to ....
#EndText
#Covered_by An element
#Justify A justification
#Link_to E_SYSTEM_INIT-001
#Maturity TBC

#EndFunction
```

Figure 11 : Preview, a step toward the final document generation

It became clear quite soon that the requirement manager was not at all specific to SAM. It led us to study its generalization as a new component. Recent experimentations done with UML/SysML proved the interest of this solution. The developments done for SAM are gradually adapted and migrated to this generic requirement manager.

4.5 OCL tools

Another need is to verify the consistency of models at any time during the modeling phase. The OCL tools offer a transparent integration with all TOPCASED modelers. Based on MDT OCL, this component allows writing OCL rules, evaluating them on models and visualizing the results in a dedicated form.

About 80 OCL rules have been defined to verify the static semantics of SAM models. They provide the end user with information and relevant to help him fixing his models. These OCL rules have been integrated into SAM plug-in and are available when users edit their model in the editor. Users must nevertheless launch manually the verification thanks to the corresponding tool bar action: this is not a live validation but a static one.

The verification results of these rules are notified into the Eclipse problem view. Corresponding decorators are displayed on graphical elements to help with identifying erroneous or incorrect elements. A double click on a problem sets the selection to the corresponding element in a diagram or in the Outline view.

4.6 Team work support

In real life, models soon become so big that several designers have to work concurrently on them. Topcased includes a feature supporting such approach. Once the main model has been initiated, it is possible to lock and export sub-tree of it. Each designer then works independently on its parts, without even needing to access the main model.

Once a designer finished its set of modifications, Topcased re-import them in the main model and unlock the corresponding sub-tree. If the interfaces between the main model and the sub-tree evolved inconsistently, they are also reconciled semi-automatically.

4.7 Airbus customisations

Of course, even if they have been designed with Airbus needs in mind, local customisations are often needed. To shorten the *Time to Desktop* of these customisations, several features have been used:

- Javascript/Python scripting engines: by providing a script-style API, these engines allow to implement model to model transformations (initialisation of the subset definition from the split up model for example), to build editing macros for specific actions. The scripts are easy to deploy and to launch from the Eclipse GUI.
- Acceleo model to text generators engine: Acceleo has been used to build quickly data extractors (flow dictionary preview, upstream requirements covering status, etc.).
- Gendoc Document generator: with the now classical template approach of Gendoc, it is possible to adapt the output format to the user's needs with no or very few Java developments.

Even with these frameworks, customisation remains a time consuming operation: dozens of scripts and templates have been developed to tune the modelling environment for specific needs.

## 5. Deployment

Functional aspect is not the only important one for a tool chain: its deployment may also be more or complex, depending on the need, the platforms, etc. At Airbus, Topcased components are not deployed in the standard bundle provided on the web site. To improve stability and functionalities, they are packaged in a specific feature that excludes unused and most experimental features – but a few ones absolutely. This bundle also includes a few specific plug-ins – generators, scripts, etc. – and some other eclipse features useful to our developments – the C and Java Development Toolkit, several VCS clients, scripts editors, etc.

This installation is provided for Solaris 8 & 10, Linux and Windows platforms. Due to the organization of our department, several deployments are then produced:

- A centralized deployment: eclipse is installed along with all the required features on a network drive. This is the preferred installation for internal users using Unix stations. Of course, a decent network bandwidth and latency is required to work this way: a 100Mb/s ethernet LAN is required.
- A full-featured eclipse archived in a zip file. To install it, it is only needed to unpack it on the local disk of the station. This is the preferred installation for external users (industrial partners) and windows platform.

Note that in an extended enterprise context, the fact that this bundle is available under the Eclipse Public License, an open source license, is a real advantage. There is therefore no need for specific licensing/lending/renting contracts.

On the performances side, our platforms are not equal faced to eclipse and Topcased. Most people would say that Windows would be the fastest one, due to the efforts of optimisation that have been done on the Java virtual machine and because eclipse is mainly developed on it, but it is not. It seems that our internal configuration slows down eclipse a lot: in some situations, eclipse can take several minutes to get launched! Moreover, some problems of stability have not been resolved, leading sometimes to crash completely the system. Solaris and Linux appear far more robust. Solaris has decent performances but it is Linux that offer the best performances (probably due to the more efficient underlying hardware, a more optimised JVM and a better X11 server).

## 6. Conclusion

After a few months of work with this toolset in real industrial conditions, we can conclude that:

- Topcased SAM now offers a functional tool chain, covering most user needs for software specification for models up to (at least) a few thousands elements.
- The model editor is seen as quite user friendly, even if for some activities (model refactoring, HTML edition, etc.) it still can be improved.
- The required support tools complete it efficiently: requirements editors, static semantics verification engine, text / model / documents generators & scripting engines.
- Test plans and soon formal properties generations improve greatly the productivity when the behavior can be described as Automata.
- The fact that the tools help is integrated is also seen as a big advantage.
- The lock/unlock approach for teamwork is still not robust enough and may be also functionally improved (CDO for example may be used to allow a finer and smoother locking grain).
- There are still some performances issues around large models and with our Windows platform. These locks are currently investigated and we hope to overcome them soon (the latest Topcased releases already include some related improvements).

In brief, the SAM toolset can now be considered as a complete and customisable solution for small & medium size projects. Its remaining flaws nevertheless have to be fixed before to deploy it to a larger scale.

## 8. References

[1] Topcased: http://www.topcased.org

[2] Eclipse Modeling Project: http://www.eclipse.org/projects/project_summary.php?projectid=modeling

## 9. Glossary

| | |
|---|---|
| *CDO :* | EMF project providing distributed shared model edition capabilities |
| *DSL:* | Domain Specific Language |
| *EMF*: | Eclipse Modeling Framework |
| *EMP:* | Eclipse Modeling Project |
| *HTML*: | HyperText Markup Language |
| *M2M*: | Model-To-Model |
| *M2T*: | Model-To-text |
| *MDE*: | Model Driven Engineering |
| *OCL*: | Object Constraint Language |
| *SAM*: | Structured Analysis Model |
| *SART*: | Structured Analysis for Real Time |
| *Topcased*: | Toolkit in OPen source for Critical Applications and SystEm Development |
| *UML*: | Unified Modeling Language |
| *VF*: | Validation Framework |