D1.2 - Assessment report on the Formose method and tools on the case studies

WP1.2 - Case Study and Tool Assessment

Abstract: This deliverable reports on the evaluation of the Formose methods and tools in the FORMOD tool, from OpenFlexo. We evaluate those tools and methods on the different use-cases of the project.

Status: Public / ~~Confidential~~
Version: Draft / ~~Review~~ / ~~Final~~

Version history

Version	Date	Contributors	Contribution
1.0	14-11-2019	Delphine Longuet	Drafting of the document
1.1	10-02-2020	Delphine Longuet	Update with new release of tool
1.2	17-02-2020	Romain Soulat	Update and corrections
1.3	14-05-2020	Delphine Longuet	Corrections

Contents

List of Figures

1 Introduction

This document reports on the evaluation of the FORMOSE methods and FORMOD tool on the Thales and Clearsy use cases. It is divided in two parts: one for the Thales use case and the other one for the Clearsy use case. Each part describes the different modelling steps of the use case following the FORMOSE methodology and comments on the usability of the tool along these steps.

Part I
Thales Use Case

2 Thales Use Case: DVOR

A DVOR (Doppler Very High Frequency Omnidirectional Range) is a radio navigation device for aircrafts. “Very high frequency" means that it operates in the band of 108 to 118 MHz. It transmits an omnidirectional signal enabling a suitably equipped aircraft to determine its position with respect to the device. It also transmits a Morse code of the DVOR station identifier and a voice signal.

A DVOR ground station uses a phased antenna array to send a highly directional signal that simulates a clockwise horizontal rotation at a rate of 30 times per second. It also sends a reference signal which is in phase with the directional signal as the latter passes magnetic north. The phase difference between the reference signal and the directional one corresponds to the angle of the aircraft relative to the DVOR station and magnetic north. The directional signal is transmitted by a circular array of omnidirectional antennas concentric about an omnidirectional antenna transmitting the reference signal.

Figure 1 shows the main high-level functional requirements extracted from the Federal Aviation Administration (FAA) requirement document, which gives the main specifications to which a DVOR ground station must comply. Thales use-case also had all the requirement levels from System requirement down to one component software requirements. We chose to only work on the highest level of requirements, client specification, as it is open to anyone and not subject to confidentiality issues. We believe that our experiments would have lead to the same results on the Thales requirements.

3 Overview of the tool

We installed the FORMOD tool on Windows 10, with Java 1.8. This corresponds to the standard Thales development environment.

When creating a project, one must give it the Formose nature to have access to the different models of the Formose methodology. The first step consists in defining the requirements, either by annotating an imported written document or by adding them manually. Then the SysML/KAOS methodology can be instantiated, in order to organise the requirements as goal diagrams. In these goal diagrams, different levels (groups of goals) must be defined, which will correspond to different levels in the domain model. Once these levels are defined, the domain model methodology can be instantiated. The domain model defines the concepts of the system as well as the relations between them. This model then serves as a basis to a B model of the system on which refinement properties can be proven.

4 Document Annotation

For the requirement elicitation phase, we worked with the Detailed Requirements section of the FAA requirement document (see an excerpt in Fig. 2).

The document annotation module of the tool allows one to import a Word document with the .docx extension in order to extract the requirements from it. Most documents from certificate bodies, or clients are PDFs, while internal documents are in .docx. Since we only had a PDF version of the DVOR specification document from the FAA, we had to reformat it as a Word document to be able to import it in the FORMOD tool. The result of its importation can be seen

Aircraft Indications. The radiated DVOR signal shall cause a standard aircraft VOR instrument to display the aircraft's angular bearing from magnetic North in degrees, referenced to the location of the DVOR.

Navigation Characteristics. The navigation characteristics of the DVOR are:

- The DVOR shall radiate a radio frequency carrier modulated by two separate 30 Hz signals which differ in phase. The phase difference between the fixed (reference) and variable phase modulation is interpreted as the navigational bearing by the VOR receiver.
- The variable phase signal shall be such that its phase, observed at a point in space, differs from the reference signal by an angle equal to the bearing of the point of observation with respect to the DVOR antenna system location.

Variable Phase Technique. The DVOR shall create the variable phase 30 Hz modulation by means of the double sideband (upper and lower) Doppler technique, i.e., by electronically simulating moving antennas.

Magnetic North Alignment. The reference and variable phase modulations shall be in phase along the reference magnetic meridian through the DVOR antenna system.

Note: The reference and variable phase modulations are in phase when the maximum value of the sum of the radio frequency carrier and the sideband energy due to the reference phase modulation occurs at the same time as the highest instantaneous frequency of the variable phase modulation.

Figure 1: Excerpt from the DVOR general requirements (FAA document)

5.1.4.2 Sideband Transmitter Requirements.

5.1.4.2.1 Sideband Transmitter.
The DVOR sideband transmitter shall provide two sideband outputs which are nominally 9960 Hz above (USB) and 9960 Hz below (LSB) the frequency of the carrier transmitter.

5.1.4.2.2 Sideband Power Output.
The sideband transmitter:
a. The sideband power shall be adequate to produce a space modulation depth of the carrier as specified in paragraph 5.1.4.3.3 at the maximum carrier power level specified in paragraph 5.1.4.1.1.1.
b. The output power shall be adjustable throughout a range specified in paragraph 5.1.4.3.3.

5.1.4.2.3 Sideband Power Output Stability and Control.
The sideband power outputs:
a. Shall be capable of automatically tracking the output power of the carrier transmitter to provide a constant depth of modulation.
b. Shall track the output power with an accuracy of within ±0.25 dB for carrier transmitter power changes of +1.0 dB to -3.0 dB.
c. Shall meet this requirement over the range of all service conditions.

Figure 2: Original Word requirement document

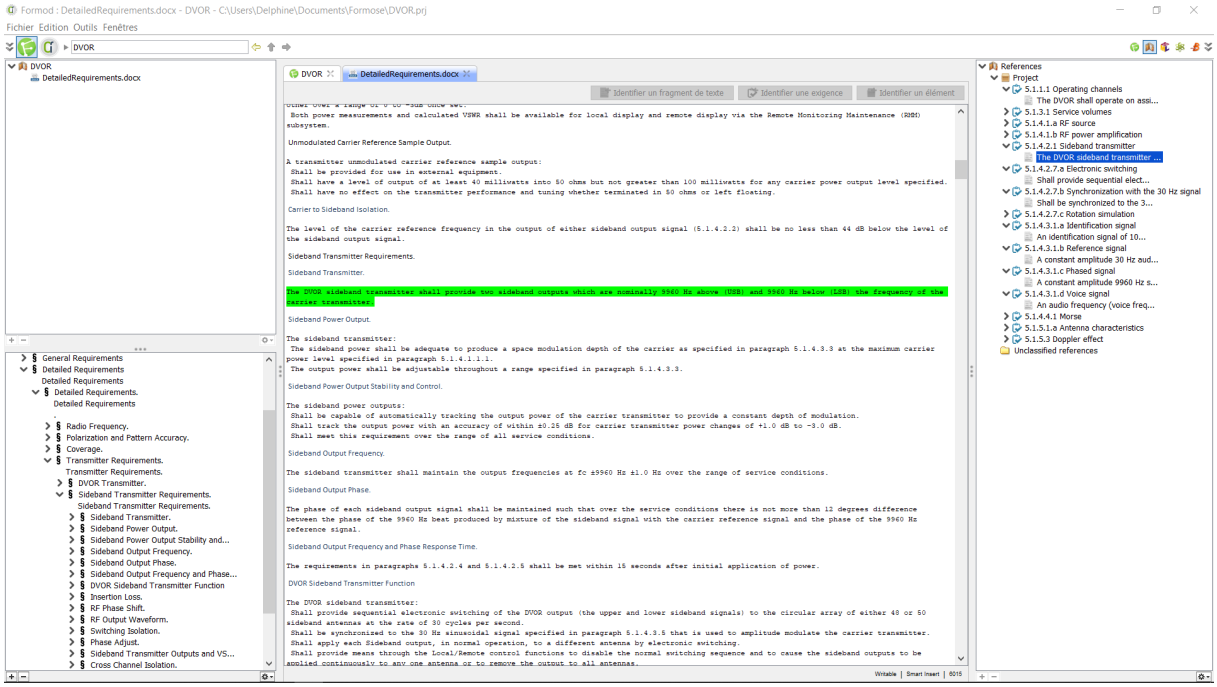

Figure 3: The document annotation interface

in Fig. 3, in the central panel of the window. The structure of the document is displayed in the bottom left panel.

A requirement document is highly structured and each requirement is numbered (see Fig. 2), so the numbering of the subsections is an important information to browse through the document. As can be seen in Fig. 3, in the current version of the tool, the importation preserves the text and the structure of the document, but looses the numbering.

Once the document is imported, it can be annotated either to extract fragments of text to be used later, to identify requirements, or to create “elements” which will give structure to the future models. The document can also be directly modified inside the tool, which may be useful for small corrections but seems to be dangerous since major changes (in the structure for example) are not synchronised with the original document.

We identified the main functional requirements of the DVOR, as can be seen on the right-hand side panel in Fig. 3.

Each requirement must be given a name and corresponds to one or several sentences of

the original text. Extracted fragments of text which are not yet identified as requirements will appear under the directory *Unclassified references*. Extracted sentences may be moved from a requirement to another or from unclassified to a requirement. Once these requirements are extracted, one can instantiate the SysML/KAOS methodology by moving to the SysML/KAOS view, in order to organise these requirements as goal diagrams.

5 SysML/KAOS Goal Diagram

When the SysML/KAOS view opens for the first time, all the requirements identified during the document annotation step are displayed in the bottom left panel. The central panel is empty. There are two ways to create a new functional goal: either by dragging and dropping one of the requirements from the bottom left panel, or by dragging and dropping an F-goal (functional goal) template from the right palette. In the latter case, a new requirement will be created and will appear with the already existing ones in the bottom left panel (see Fig. 4).

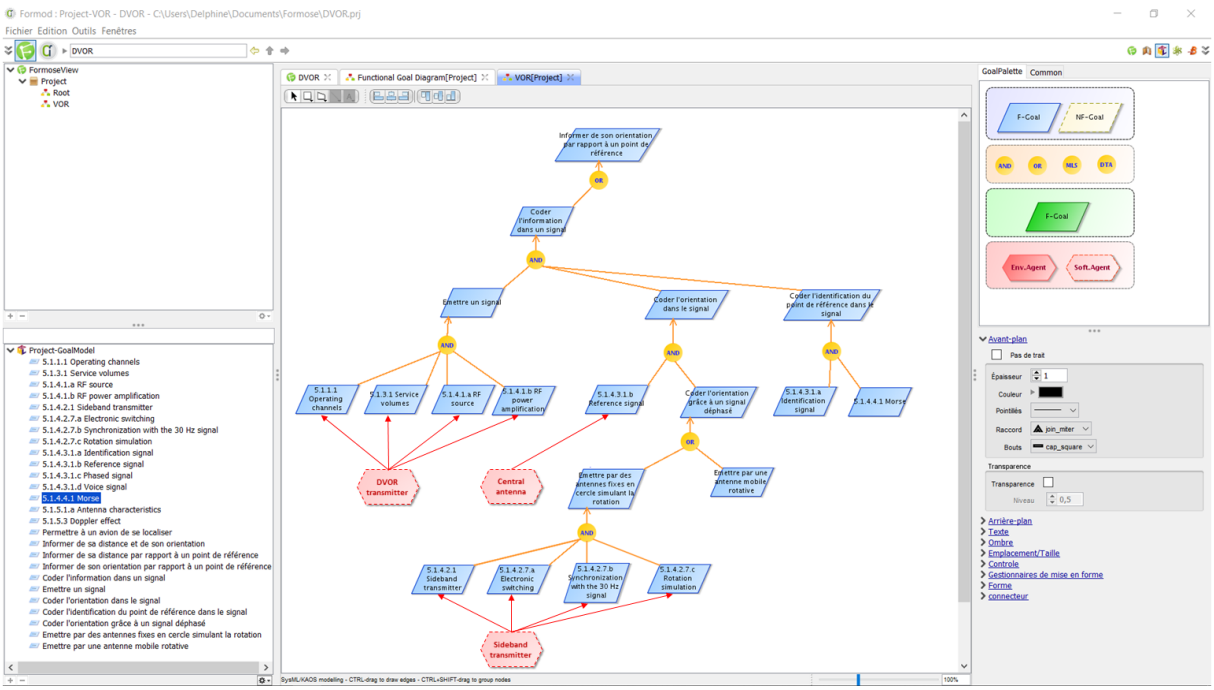

Figure 4: Functional goal diagram of a DVOR

One can see that, in our goal diagram of the DVOR, high-level goals at the top of the diagram were created by hand, while the requirements identified in the document appear as subgoals, at the bottom of the diagram. This is due to the fact that the requirements are already rather detailed and sometimes technical in the FAA document. There does not exist a more general document justifying these technical choices since a DVOR is a well-known device which answers to established and well-understood aircraft navigation rules.

We also note that goal diagrams aim at providing several alternatives for some refinement (with the use of the “or” connector). In this use-case, most of the choices have already been made by the client, probably during an earlier phase. Our goal diagram is mostly composed of “and” and “milestone” connectors. We believe that this will be the case in a lot of systems

where the design is highly influenced by previous products and standards. We can see how, in a more agile environment, choices might exist, maybe even partially implemented before justifying which one is taken. Overall, thinking in terms of goals makes the engineers work in a different mind set. Goals should express “what” or “why” the system is being built, and not the implementation. Engineers tend to make choices at the early levels of design by coming with solutions.

The goals at the bottom of the diagram may be assigned to agents, being either part of the environment or part of the system itself. These goals may themselves be refined into subgoals in other goal diagrams, thus focusing on the corresponding agent. Requirements of the bottom left panel may be dragged and dropped several times in different diagrams if necessary, for example if one of the bottom requirement must be the root of another goal diagram refining it. The different goal diagrams are shown in the top left panel.

Besides functional goals, non functional requirements may also be structured as goal diagrams. Non functional requirements are most of the time linked to functional requirements for the part of the system designed to realise or ensure these non functional requirements. For example in the case of the DVOR, in order to ensure the quality of the emitted signal (which is a non functional requirement), a monitoring device is added to the main system, this device having its own functional requirements to be able to meet its purpose (see Fig. 5).

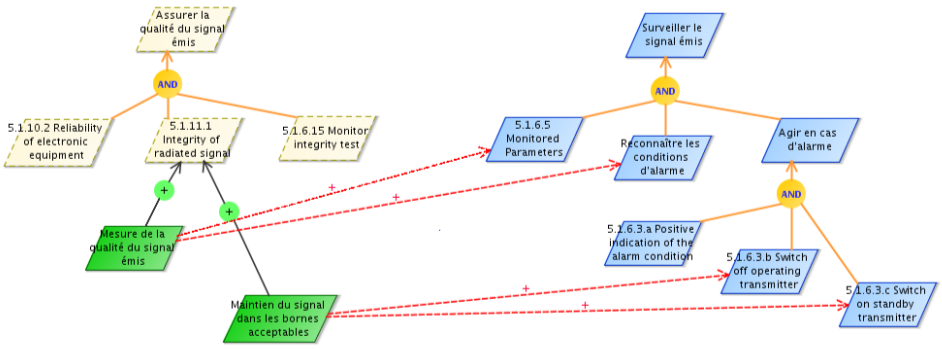

Figure 5: Non functional goals and their corresponding functional goals

When several ways of meeting a non functional requirement are possible, contribution goals may be added to explicitly show positive or negative effect of a functional goal to a non functional one. For example in a VOR, the choice of a fixed circular array of antenna (DVOR) instead of a rotating antenna (CVOR) comes from the non functional requirement called Solid-state design, stating that the device shall not have any moving parts. The corresponding part of the diagram may be seen in Fig. 6.

The interface of this view makes the creation of goal diagrams rather easy thanks to drag-and-drop. Each new requirement created by hand is also created in the Document Annotation view, which allows to link it to a fragment of text in the document even after the requirement identification phase. In the other way round, if a new requirement is identified, it is added to the list of requirements and may be used to create a goal in the diagram.

The functional goal diagram will serve as a basis to the creation of a domain model. To be able to give structure to the domain model and to the future B model, one must first identify *levels*, which are layers of goals. To each level of goals will correspond a part of the domain model that focuses on these goals, by defining the concepts introduced by these goals as well

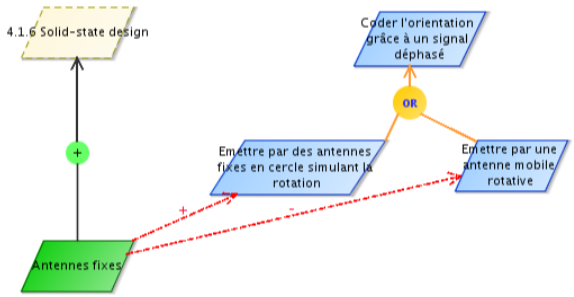

Figure 6: Non functional goals influencing choices between functional goals

as the relations between them and with the concepts introduced at higher levels.

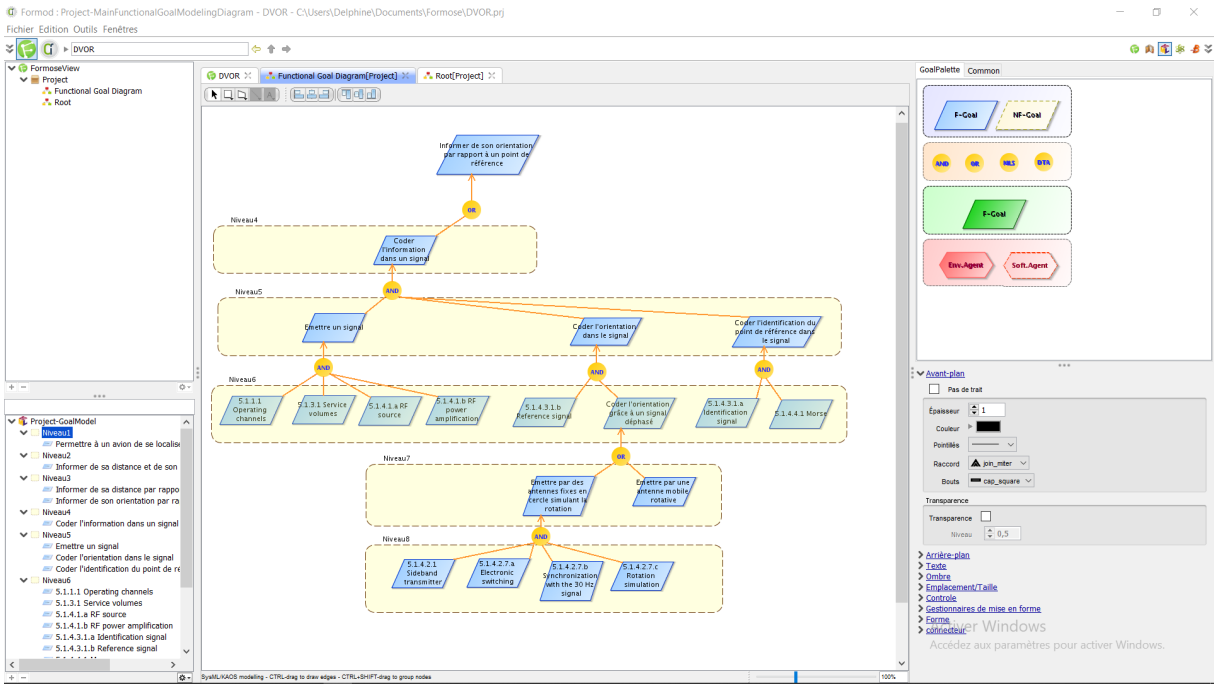

Figure 7: Definition of levels into the goal diagram

As we can see in Fig. 7, our goal diagram has 8 levels, that will correspond to as many levels in the domain model. Unfortunately, the methodology only allows levels to contain goals located at the same depth of the diagram, which we feel is a too strong restriction in practice. Furthermore, realistic models might be composed of several goal models, which makes impossible the definition of levels that would gather goals of different goal models.

6 Domain Model

Once levels are defined in the goal diagram(s) of the SysML/KAOS view, one can instantiate the Domain Model methodology.

For each level, the concepts introduced at this level of abstraction must be created as con-

cepts in the domain model, and their relations modelled as associations. The interface allows one to create objects by drag-and-drop from the right palette (see Fig. 8), even for attributes and formulas. The red objects are concepts, the yellow ones are associations and the green ones are individuals.

The bottom left panel displays for each level the objects created at this level, and the objects defined at higher levels. Levels inherit from one another, so objects introduced at level i may be used at level j for $j > i$, by dragging and dropping them from the bottom left panel. This enables to create relations between objects defined at different levels, as we can see in Fig. 8.

- A concept may be refined into subconcepts: SidebandSignal defined at level 5 inherits from Signal defined at level 4.
- Individuals of a concept may be created: SidebandSignal_0 defined at level 8 is an instance of SidebandSignal defined at level 7.
- Associations may be created between existing concepts: EmittedSignal defined at level 8 links the concept of Antenna defined at level 7 and the concept of Signal defined at level 4.

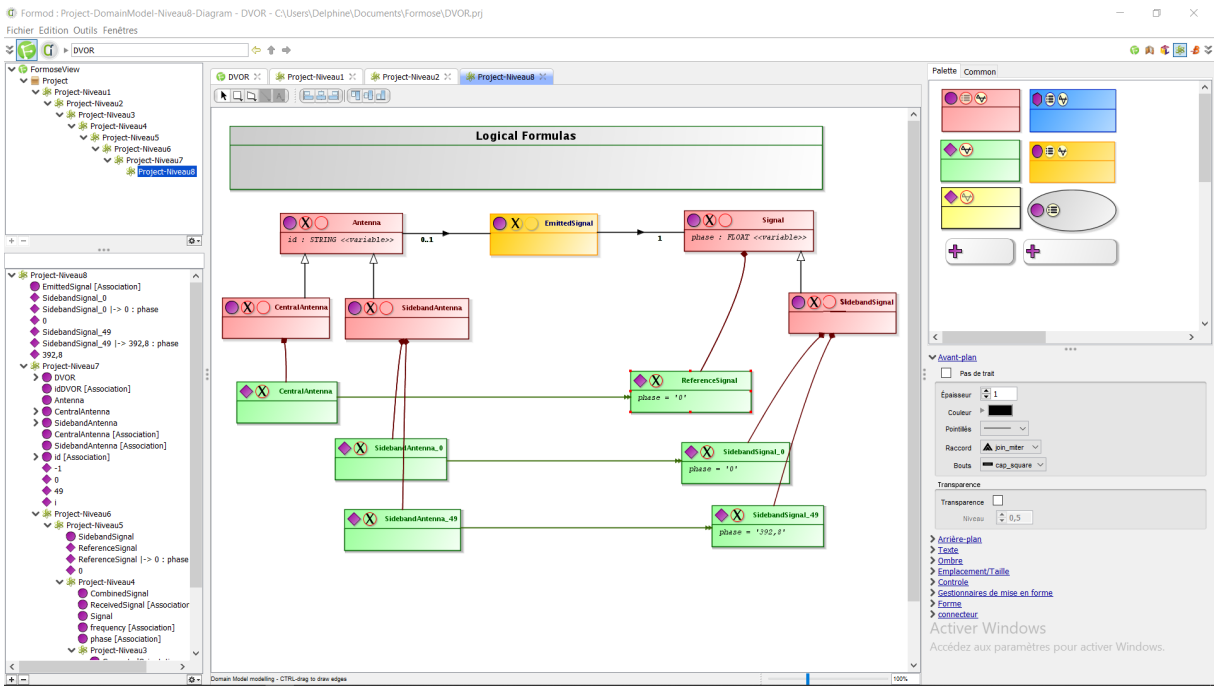

Figure 8: Level 8 of the DVOR domain model

The interface of this view makes it rather easy to create domain models, even if multiple options are often proposed in the contextual menus, that are not always easy to understand. Besides, it would be nice to propose some help to the user to write syntactically correct formulas (that must be written in B already at this step).

7 B Model

At the very end of the chain of models, the structure of a B model can be generated from the previous models (see Fig. 9). This view displays at the same time the goal diagram, a level of the domain model, and its corresponding B context (on the left) and B machine (on the right). What remains to the user is to write the formulas describing the behaviour of the system at each refinement step. These formulas, and the rest of the B model must be written by hand.

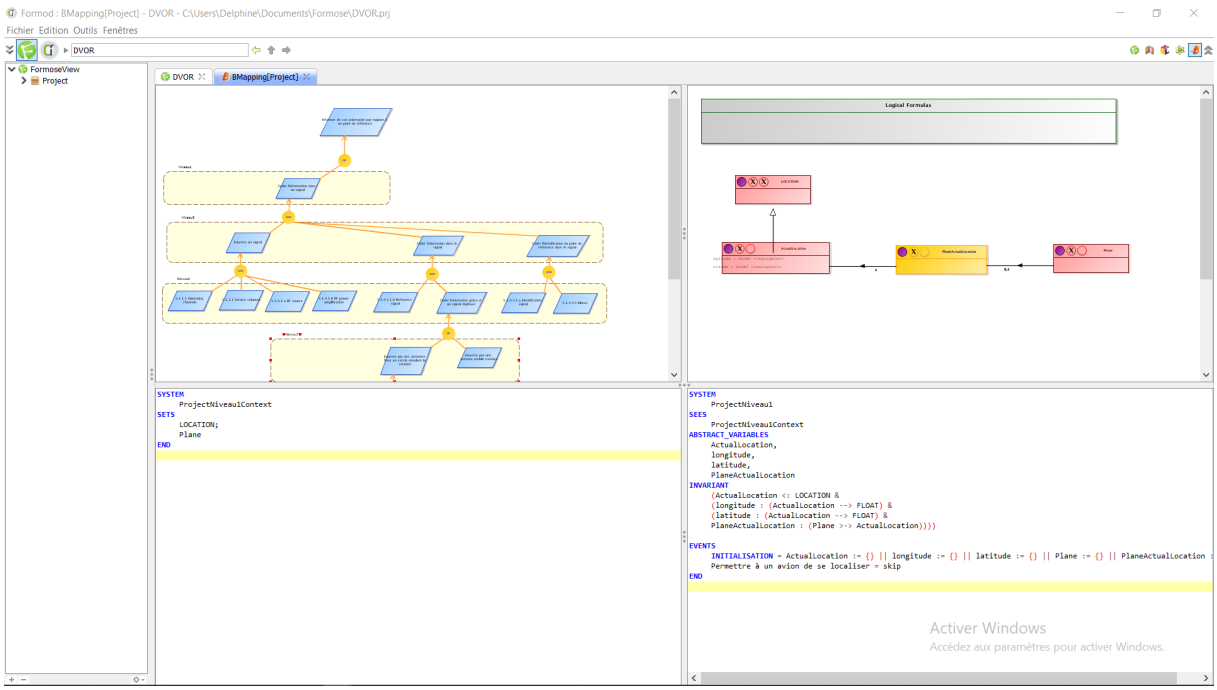

Figure 9: Result of the export to B

The generated B model can be opened in AtelierB to be completed, so that the refinement steps can be proven. The model can also be directly modified in the FORMOD tool which uses AtelierB as backend for syntax checking. This enables the user to keep every model in the same tool and have a global view of the different models. Furthermore, any modifications made to the model in FORMOD or in AtelierB is synchronised with the other tool, which makes the entire process completely integrated.

Once the B model is complete, the proof is done in AtelierB as any B model would be proven. This step would not be done by the system, or software architect in a company as Thales.

8 Conclusion

The FORMOD tool in its current state can be used to create models following the FORMOSE methodology, from a requirement document to a B model that can ultimately be proven. It provides a nice interface and a rather easy way to draw diagrams by dragging and dropping objet templates or objects existing from a previous model. We demonstrated its usability on the realistic use case of the DVOR.

The main problem we encountered with the tool is the loss of information when loading an existing project. All the objects of the project still exist graphically but their nature and the data attached to them seem to be lost, which prevents from making essential modifications to the project.

From a methodological point of view, the current state of the tool allows one to create the whole chain of models from the document annotation to the B model, but does not really allow to step back from a model to a previous one. For instance, once a domain model is instantiated, it is possible to add new levels in the goal diagram but no other modification is taken into account (adding or deleting goals, reorganising them, or modifying levels). As another example, once the B model is created, it is no longer possible to modify the domain model. To be used on realistic projects, the tool needs to provide more flexibility to changes, or at least inform the user when a change she wants to make may break the global consistency between models.

In its current state, the FORMOD tool is an acceptable prototype which provides all the functionalities necessary to follow the FORMOSE methodology. Yet, we think it needs to show more robustness to be considered in an industrial setting. Moreover, we feel that during the goal elicitation, and domain models definition, one must be aware of the B model that will be generated. As it stands, it would be an issue to deliver this product and methodology to Thales system/software architects. We believe that this drawback could be addressed in further development.

Part II
Clearsy Use Case

9 Clearsy Use Case

SATURN is a computer network allowing input acquisition, data processing and control output developed by Clearsy. SATURN is made up of two main components: a concentrator and input/output (I/O) agents. These components are connected through a network having a ring topology structured under a master-slave communication model where the concentrator has the master role and the input/output agents the slave role. The communication protocol is started by a synchronization frame sent by the concentrator to the I/O agents; the synchronization frame contains the state of the outputs to be applied by output agents. When the synchronization frame is received by an output agent, it sets its output according to the requested values in this frame. When the synchronization frame is received by an input agent, it sends its input values to the concentrator. The concentrator uses the inputs received in response of a synchronization frame to calculate the output to be sent in the next synchronization frame.

In this report we present how Formose methodology has been applied in the abstract requirements of the SATURN computer network taking special attention to goals formalization. A top-down approach is illustrated starting with the presentation of the main goal of the system in section 10. Then in section 11 we descend in the refinement to show how the atomicity of the main goal is refined; in particular we make goal and data refinements through Goal and Domain Diagrams and we use the Proof Obligations to validate these refinements. After the refinement of atomicity, in section 12 we present how the Formose refinements are used to show some alternatives in the communication protocol and then we show another refinement of the choices made in the SATURN network. Finally, in section 13 we give our conclusions on this Case Study.

10 The Main Goal

The Main Goal of the SATURN network is to compute outputs according to the state of its inputs. The relationship between inputs and outputs is modeled by a total function, assigning a single output to inputs combinations.

We formalise goals in Goal Diagram as shown in figure 10. In this figure, below the name of the goal, we have the expression $guard\ LEADSTO\ postcondition$. This expression uses the *leads to* operator ($\rightsquigarrow$) to denote a state transformation from a state where the *guard* holds to a state where the *postocodition* holds. In this way, $guard$ and $postcondition$ are predicates over the system to be specified.

Figure 10: Formalizing Goals

In order to define the state of the system, we use the Domain Diagram depicted in figure 11. The concepts T_IN et T_OUT denote the set of values for inputs and outputs respectively. These concepts are related through the association FB modeling a boolean function, where input values are associating to output values. Two particular individuals of T_IN and T_OUT are denoted by *constants* $i0$ and $o0$ respectively; these constants are related through the FB function as illustrated by the logical formula of the Domain Diagram.

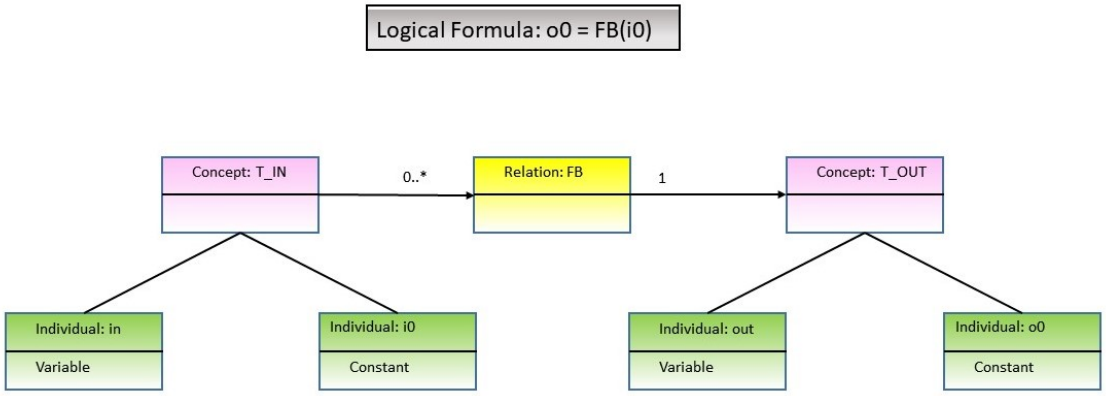

Figure 11: Domain Diagram corresponding to the Main Goal

The Domain Diagram in figure 11 is captured in the Formod tool through the Domain View. The figure 12 shows the Domain View with the Domain Diagram of the Main Goal.

The state of the system is modeled by two other individuals of T_IN and T_OUT having a *variable* kind: in and out respectively. Having this Domain Diagram defined, we formalize the Main Goal in figure 13. The Main Goal states that the system must be able to pass from any state ($true$ guard) to a state where the output out variable gets the value corresponding to $FB(in)$.

Having defined the Goal and Domain models, the Formod tool generates two Event-B components: one for the system context, issued from the Domain Diagram and another one, for the dynamic behavior issued from the Goal Diagram, containing the template of the Event-B events. In this case, the template contains only one event named *Saturn* with $\mathtt{skip}$ as its body. This template is completed with the $Saturn$ goal of figure 13 as follows:

$$
\begin{array}{l}
Saturn = \\
\quad \text{SELECT} \\
\quad\ 0 = 0 \\
\quad \text{THEN} \\
\quad\ out : (out = FB(in)) \\
\quad \text{END}
\end{array}
$$

It must be noted that the post-condition of the *Saturn* goal is translated into Event-B as a "Becomes Such That" substitution, indicating that the *out* variable becomes equal to $FB(in)$. An alternate way to write the substitution in Event-B is by using a deterministic assignment as $out := FB(in)$. The AtelierB tool generates and discharges all Proof Obligations involving mainly typing constraints.

In the next section, the Main Goal $true \rightsquigarrow out = FB(in)$ is refined to consider the distributivity of input/output agents.

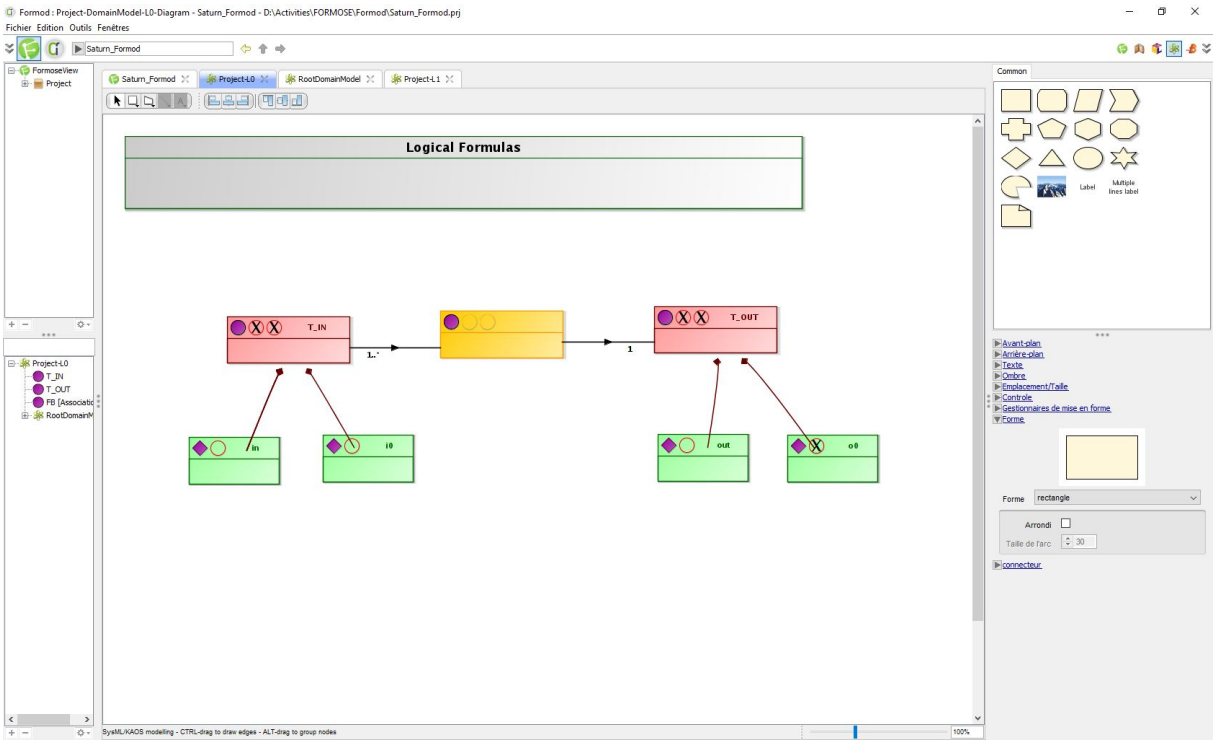

Figure 12: Domain Diagram of the Main Goal in the Domain View of Formod

Figure 13: Main Goal of SATURN Network

11 Refining Goal and Data Atomicity

In this section we introduce the notion of input and output agents. The introduction is made in three parts. First, we introduce the input agents, stressing the importance of proof obligations to validate the refinements. Then we introduce the output agents to complete the picture of data transfer in the SATURN network. The final part shows a data refinement technique to discretise the I/O agents.

11.1 Introducing the Input Agents

In this section the Domain Diagram is modified to introduce two variables which are individuals of the T_IN concept. The Domain Diagram of the first refinement level is presented in figure 14.

The idea behind this raffinement is to distinguish the memory in input agents and the memory in the concentrator. Data acquired by captors in the input agents are stored in the *remote* variable in_r. This data is transmitted in a certain way to the concentrator and then stored in

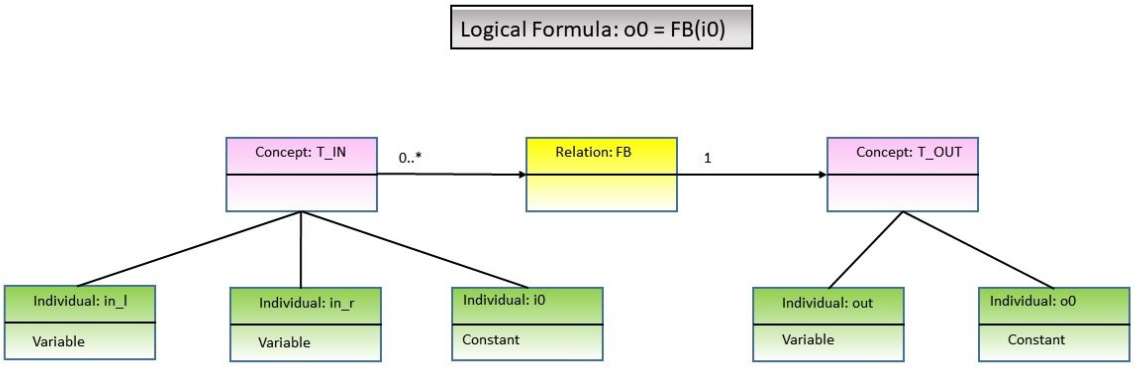

Figure 14: Domain Diagram introducing Local and Remote Input Variables

the *local* variable in_l.

It must be noted that abstract variable in in the Domain Diagram corresponding to the Main Gaol does not appear in the Domain Diagram of figure 14. In fact, we are refining the abstract variable in by a concrete variable. In this case we have two new variables in the refinement: in_l and in_r and we need to add a new *Logical Formula* in the the Domain Diagram of the figure 14 to relate the abstract variable in with a concrete variable. The Logical Formumla must have the form:

$$in = x$$

where we must decide to replace x by in_l or in_r. This choice will be treated in the next paragraphs, but we need to present the Goal Diagram of this refinement first.

For the sake of completeness, we make two attempts to present the Goal Diagram of the First Refinement Level. The first attempt uses an AND refinement of the main goal. In this first attempt, we present the two ways of gluing the abstract variable in to the concrete variables, but we show how this AND refinement fails to be proved. In the second attempt, we show a MILESTONE refinement of the main goal which is correctly proved.

11.1.1 AND-Refinement of the Main Goal

The figure 15 shows the Goal Diagram of an AND refinement of the Main Goal. Two new sub-goals are introduced *Get* and *Control*. The *Get* goal denotes the transfer of data from the Input Agent to the Concentrator whereas the *Control* goal denotes the output computation by the Concentrator.

In the Formod Tool, Goal Diagrams are made in the SysML/KAOS View. In that View, Goals are depicted and then they can be related through the different kinds of refinement. In figure 16 we show an screeshot of the SysML/KAOS View containing the AND Refinement of the Main Goal of the SATURN Network.

According to the AND Refinement, these two sub-goals must be executed at the same time. Semantically, the variables are update in *parallel* as described by the following generalized substitution :

$$in\_l,out := in\_r,FB(in\_l) \quad (1)$$

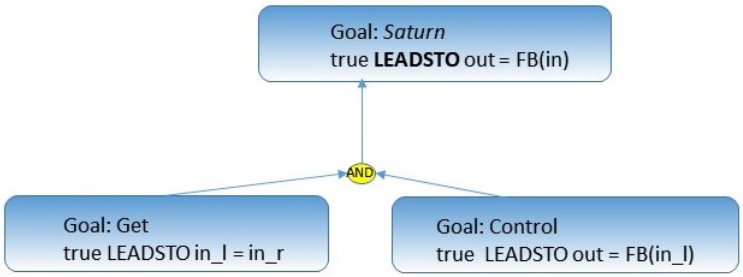

Figure 15: AND Refinement of the Main Goal of SATURN Network

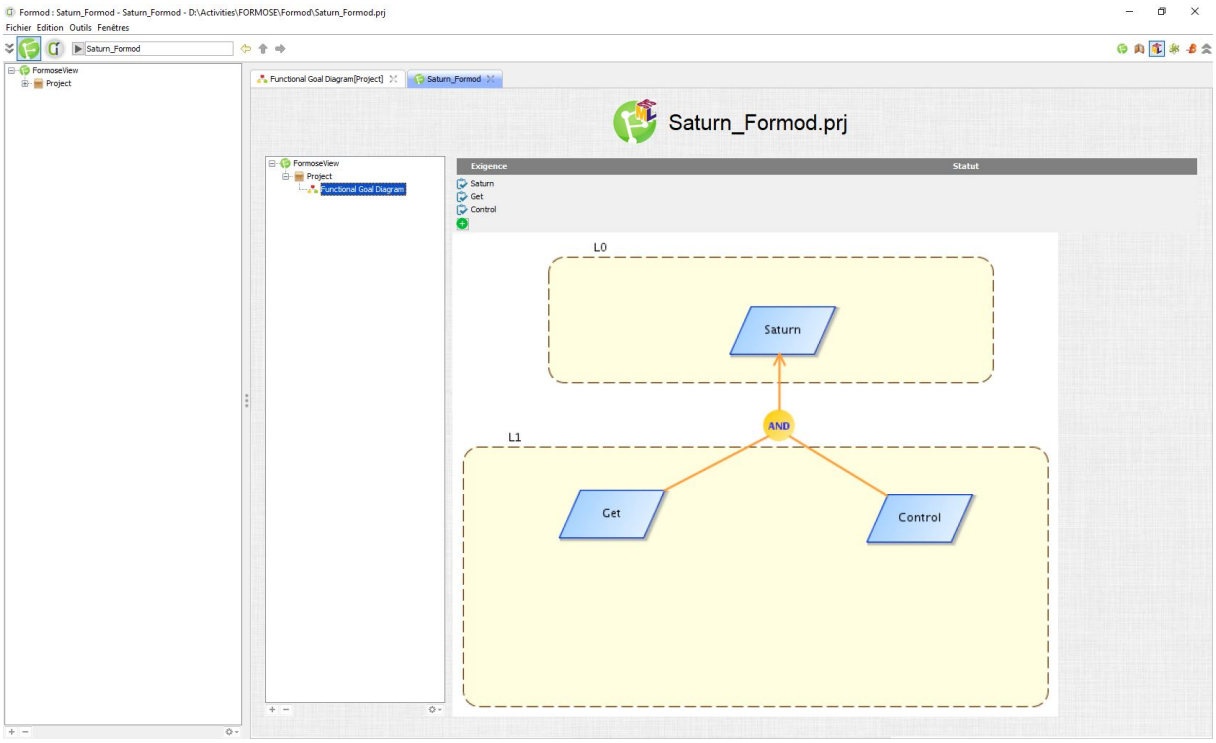

Figure 16: Main Goal's Refinement in the SysML/KAOS View of Formod

In this kind of substitution, the expressions on the left hand side of the assignment are first evaluated and then, the assignments are done at the same time.

As we indicated in previous paragraphs, we need to specify a Logical Formula relating the abstract variable in with a variable of the Domain Model in figure 14. As the postcondition of the *Control* goal states $out = FB(in\_l)$ and we need to prove that the postcondition of the *Saturn* goal holds, it seems natural to have the following Logical Formula:

$$in = in\_l$$

In this way we can deduce the postcondition of *Saturn* goal: $out = FB(in)$.

The Logical Formula in the Domain Diagram is translated as a gluing invariant in the generated Event-B Model. Moreover, the modified generated template contains the following events:

Get `ref_and` $Saturn$ =
SELECT
$0 = 0$
THEN
$in\_l : (in\_l = in\_r)$
END

$Control$ `ref_and` $Saturn$ =
SELECT
$0 = 0$
THEN
$out : (out = FB(in\_l))$
END

The Proof Obligations for this refinement, among others, ask for the proof of the gluing invariant preservation; as it is defined by the Logical Formula $in = in\_l$, the PO to be proved is:

$$\begin{array}{l} out' = FB(in\_l) \ \wedge \\ in\_l' = in\_r \\ \Rightarrow \\ in\_l' = in \end{array}$$

In this Proof Obligation out' denotes the value of the out variable *after* execution of event $Control$. In the same way, in_l' denotes the value of in_l variable after execution of Get event. Using the gluing invariant $in = in\_l$, which holds before the execution of goal, we can transform the current goal as follows:

$$in\_l' = in\_l$$

This derived goal holds if after execution of Get event, the value of the in_l variable does not change, that is, the new value of in_l is equal to its previous value. We have no way to prove such that statement, and therefore we cannot claim that the Mail Goal is correctly refined.

An informal way to realize the unfeasibility of the proof is to see that the abstract variable in is not modified by the Main Goal, whereas the in_l variable of the AND refinement, related to in in the gluing invariant, is modified. This observation can be used to fix the Logical Formula by relating the abstract variable in with the concrete variable in_r which is not modified in this refinement. Therefore we tray to prove the refinement with the following Logical Formula:

$$in = in\_r$$

Using this formula as gluing invariant, the AtelierB is not able to discharge all the Proof Obligations again. This time the unproved PO concerns the preservation of the implicit gluing invariant:

$$out\_abstract = out\_concrete$$

where $out\_abstract$ denotes the out variable used in the *Saturn* goal and $out\_concrete$ the out variable in the $Control$ goal. The generated PO is the following:

$$\begin{array}{l} out' = FB(in\_l) \ \wedge \\ in\_l' = in\_r \\ \Rightarrow \\ FB(in) = out' \end{array}$$

In the left hand side of the equality in the consequent of this PO we can distinguish the value assigned to $out\_abstract$ as stated in the $Saturn$ goal. In the right hand side of the equality we have the value of $out\_concrete$ after execution of the $Control$ goal. From the antecedents we also observe that the value of out' is equal to $FB(in\_l)$ as specified by the $Control$ goal. Therefore, the equivalent goal to prove is

$$FB(in) = FB(in\_l)$$

The only way to prove this goal is to deduce $in = in\_l$ after execution of the *Control* goal. However there is no way to prove this equality; in_l is updated with the value of in_r as specified by the *Get* goal, but this update is made after evaluation of the right hand side of the assignment described in 1. When the right hand side is evaluated, the value of the in_l variable used to compute the output is not yet equal to in_r. This last observation leads us to conclude that we cannot specify the behavior of the abstract system by imposing a parallel execution of the goals postconditions, but we need a sequential execution of the postconditions as required in a MILESTONE refinement.

11.1.2 MILESTONE-Refinement of the Main Goal

The Goal Diagram of the MILESTONE refinement of $Saturn$ goal is similar to the Goal Diagram of figure 15; the difference between these two diagrams is the type of node joining the refined goals with the abstract one and the convention of the execution order of the refined goals. In fact, to denote a MILESTONE refinement, the node is tagged by the “MLS” acronym and the execution order of the refined goals is read from left to right. Therefore, in this refinement the *Get* goal is executed first and then the goal *Control* is executed. This diagram specifies that the local variable in_l must be updated with the value of the remote variable in_r and then the local variable can be used to compute the output variable through the boolean function.

The body of goals *Get* and *Control* does not change with respect to the bodies in the AND refinement, showed in the paragraphs above. The only change is the reserved word `ref_and` which is changed by `ref_milestone`. This time, the AtelierB has no problem in the automatic proof of the generated Proof Obligations.

In order to complete this first refinement level, we introduce the notion of output agents in the next paragraph.

11.2 Introducing the Output Agents

Output Agents notion is introduced by adding two new variables in the Domain Diagram presented in figure 17, as we have done in the introduction of Input Agents. The out_l variable is the local variable used by the Concentrator to compute the output of the system and the out_r variable is used by the Output Agents to command their output devices. In the Domain diagram

we specify by a Logical Formula that the abstract output variable out is glued with the concrete variable out_l.

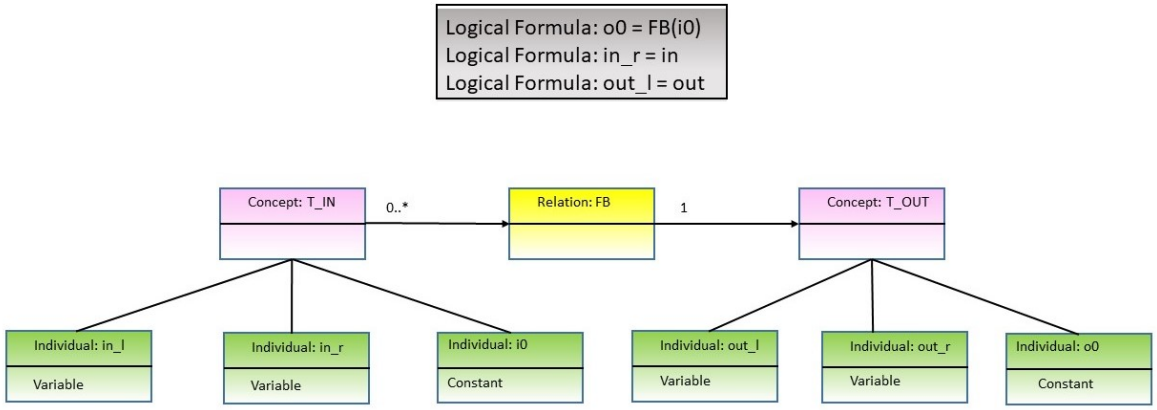

Figure 17: Domain Diagram introducing Local and Remote Output Variables

This refinement is completed by the Goal Diagram of figure 18. In this diagram the *Control* goal is MILESTONE refined by two subgoals: *Computation* and *Put*. *Computation* goal is executed after execution of *Get* goal to compute the outputs from the state of the in_l and *Put* goal is executed after computation of *Computation* goal to specify that the value of the computed local output variable out_l must be transfered to the Outputs Agents to update their local variables out_r.

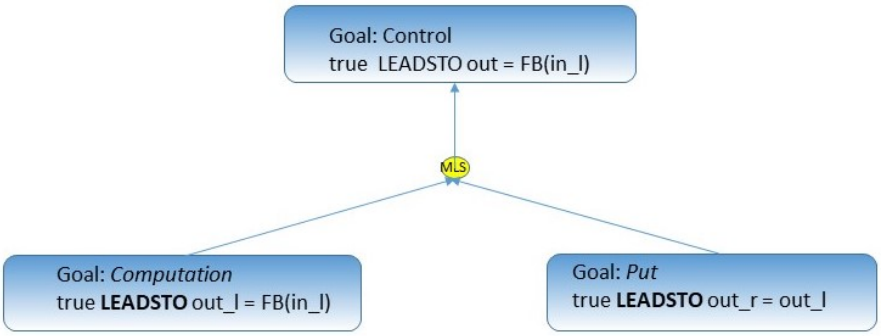

Figure 18: MILESTONE Refinement of the Control Goal

From the Domain and Goal Diagrams presented above, the Formod tool is able to generate a Event-B template. The two events of the template corresponding to the Goal Diagram are

modified as follows to formalize the two subgoals:

Computation ref_milestone =
SELECT
$0 = 0$
THEN
$out\_l : (out\_l = FB(in\_l))$
END

Put ref_milestones =
SELECT
$0 = 0$
THEN
$out\_r : (out\_r = out\_l)$
END

As in the previous refinement, the AtelierB has no problem with the automatic proof of the Proof Obligations and we are able to finish this refinement level by refining the atomicity of the variables as explained in the following paragraphs.

11.3 Data Refinement

The idea behind this refinement is to individualize input and outputs agents and their associated data variables. This individualization is formalized through the excerpt of the Domain Diagram depicted in the figure 19. In order to distinguish the agents we introduce the concepts MI and MO to denote the set of all input and output agents respectively. A given configuration of SATURN has a fixed number of input and outputs agents. Therefore, we define the concepts $agents\_in$ and $agents\_out$, which are derived from MI and MO, to denote the set of input and output agents in a given SATURN configuration. In this refinement we suppose that input and output data manipulated by these agents is of type $BOOL$. Thus, we defined the concept VIN as the set of total functions between $agent\_in$ and $BOOL$; this definition is specified by the logical formula

$$VIN = agents\_in \rightarrow BOOL \tag{2}$$

In a similar way the $VOUT$ concept is defined by

$$VOUT = agents\_out \rightarrow BOOL \tag{3}$$

In this way, VIN and $VOUT$ are seen as concrete data types used to type the variables of the agents. The association VFB models the computation function taking input data of type VIN and producing output data of type $VOUT$. Moreover the VIN concept is used to define the input variables s_in_r and s_in_l and the initialization value s_i0. The s_in_l variable denotes the input variable of all input agents as seen by the concentrator, whereas s_in_r denotes the remotes variables updated by the input agents from data coming from their input devices. In a similar way, the $VOUT$ concept is used to define the individuals s_out_l and s_out_r. s_out_l denotes the output variable as seen by the concentrator containing the data computed through the VBF function with the local input data s_in_l whereas s_out_r represents the remote variables of the output agents used to command output devices.

The attributes vec_to_in and vec_to_out of concepts VIN and $VOUT$ in the Domain Diagram are used in the correctness proof of this data refinement. vec_to_in is a total injection between the concrete data type VIN and the abstract one T_IN ($VIN \rightarrowtail T\_IN$). vec_to_out is a total function between the concrete data type $VOUT$ and the abstract one T_OUT ($VOUT \rightarrowtail T\_OUT$). The injective characteristic of these functions is specified in the Domain Diagram by a cardinality of 0..1 in the domain side of the function and a cardinality of

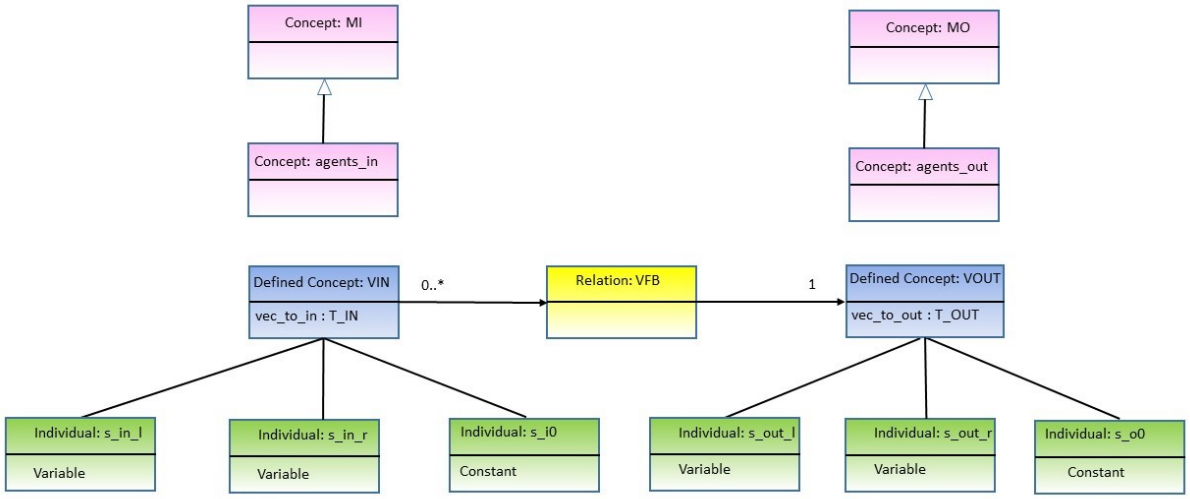

Figure 19: Domain Diagram for Data Refinement

1 in the range side. These functions allow us to complete the definition of VFB association and define gluing relations between abstract and concrete variables as defined by the following logical formulas:

$$VFB = (vec\_to\_in\ ;\ FB\ ;\ vec\_to\_out^{-1}) \quad (4)$$
$$in\_l = vec\_to\_in(s\_in\_l) \quad (5)$$
$$in\_r = vec\_to\_in(s\_in\_r) \quad (6)$$
$$out\_l = vec\_to\_out(s\_out\_l) \quad (7)$$
$$out\_r = vec\_to\_out(s\_out\_r) \quad (8)$$

To complete this refinement level, we show the Goal Diagram of the data refinement in the figure 20. The diagram illustrates the refinement of the current three sub-goals: *Get*, *Computation* and *Put*. The corresponding refined goals have been renamed with the postfix _S to denote the synchronous update of the memory of all input and outputs agents.

Events generated from the Domain and Goal diagrams are completed as follows:

S_Get ref
Get =
SELECT
$0 = 0$
THEN
s_in_l
$: (s\_in\_l = s\_in\_r)$
END

$S\_Computation$ ref
$Computation$ =
SELECT
$0 = 0$
THEN
s_out_l
$: (s\_out\_l = VFB(s\_in\_l))$
END

S_Put ref
Put =
SELECT
$0 = 0$
THEN
s_out_r
$: (s\_out\_r = s\_out\_l))$
END

The Proof Obligations generated by the AtelierB tool were mainly discharged by the automatic prover. However the preservation of the gluing relation 7 by the *Computation_S* goal had

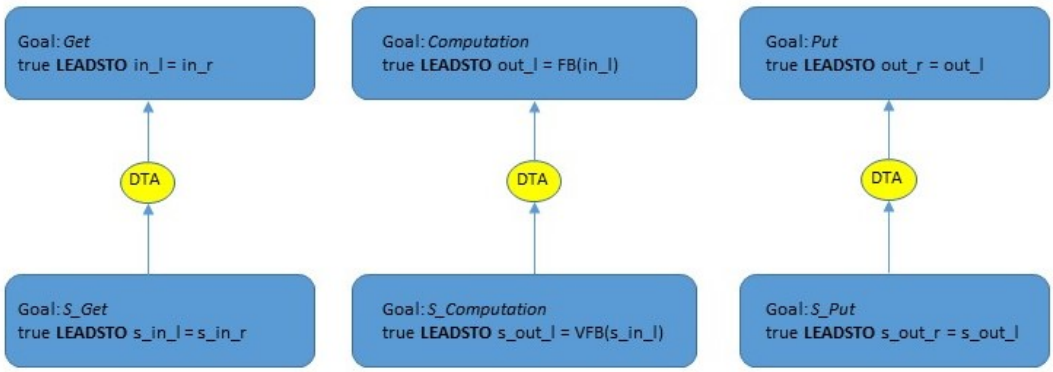

Figure 20: Data Refinement of *Get*, *Computation* and *Put* Goals

to be manually proved. The generated Proof Obligation is the following:

$$s\_out\_l' = VBF(s\_in\_l)$$
$$\Rightarrow$$
$$FB(in\_l) = vec\_to\_out(s\_out\_l')$$

The consequent of this proof corresponds to the gluing relation 7 where out_l is replaced by $FB(in\_l)$ according to the postcondition of the abstract goal *Computation* whereas s_out_l is replaced by $VBF(s\_in\_l)$ according to the postcondition of the concrete goal *Computation_S*. By using forward functional composition, this consequent can be set in an equivalent form as follows:

$$FB(in\_l) = (VBF\ ;\ vec\_to\_out)(s\_in\_l)$$

Using the Logical Formula 4 and taking into account that

$$vec\_to\_out^{-1}\ ;\ vec\_to\_out = \mathrm{id}(T\_OUT)$$

the predicate to prove becomes:

$$FB(in\_l) = (vec\_to\_in\ ;\ FB\ ;\ \mathrm{id}(T\_OUT))(s\_in\_l)$$

This last predicate can be rewritten as follows considering the fact that $r\ ;\ id(\mathrm{ran}(r)) = r$ for any relation r:

$$FB(in\_l) = FB(vec\_to\_in(s\_in\_l))$$

Finally, using the gluing relation 5, we prove this last statement.

12 Refining The Communication Protocol

In this section we explore some alternatives to specify the communication protocol between input/output agents and the concentrator. First, we begin the alternatives with the choice between a *centralized* or *distributed* communication implementation. Then, for each kind of implementation, we specify that communication can be done by a *point to point* communication or by a *broadcast*.

The kind of implementation and the type of communication between agents an concentrator is specified by two new variables used in the conditions of the guards to be specified. However, the postconditions are not modified with respect to their specifications in the refined goals of the Goals Diagram in figure 20. For the sake of conciseness, we do not illustrate the Domain and Goal Diagrams of these refinement and they are informally presented.

This section terminates with the specification of choices of coordination and communication type made in the SATURN network.

12.1 Implementation Type

Communication implementation between agents and concentrator is specified through the $Implementation$ variable having the enumerate type $\{Centralized\ , Distributed\}$. When the implementation is centralized, the concentrator requests for input to input agents and sends data to output agents, while in a distributed implementation, input agents take the initiative to send input data to the concentrator and output agents request output data to the concentrator.

The two possible implementations of the *S_Get* subgoal is specified by the follwoing OR refinement:

Distributed_Get ref_or *S_Get* $\widehat{=}$

$$Implementation = Distributed \rightsquigarrow s\_in\_l = s\_in\_r \quad (9)$$

Centralized_Get ref_or *S_Get* $\widehat{=}$

$$Implementation = Centralized \rightsquigarrow s\_in\_l = s\_in\_r \quad (10)$$

whereas the two possible implementions the *S_Put* goal is given by the following OR refinement:

Distributed_Put ref_or *S_Put* $\widehat{=}$

$$Implementation = Distributed \rightsquigarrow s\_out\_r = s\_out\_l \quad (11)$$

Centralized_Put ref_or *S_Put* $\widehat{=}$

$$Implementation = Centralized \rightsquigarrow s\_out\_r = s\_out\_l \quad (12)$$

The Proof Obligations of an OR refinement allow us to ensure that any refined goal refines, in an Event-B way, the abstract goal and that the guards of the refined goals are exclusive. In this case, the AtelierB tool has no probem with these Proof Obligations as the postconditions of the refined goals are the same that the abstract postcondition and moreover, the refined guards are strengthened.

12.2 Communication Type

Communication type is specified through the $Communication$ variable of type $\{$*Broadcast*, $PointToPoint\}$. In a communication of type broadcast (BRC) the concentrator has data channels shared among the input and output agents while in a communication of type point to point (P2P), the number of data channels is proportional to the number of agents.

Thus, each goal g in 9..12 is OR refined by two goals g_1 and g_2 where the postconditions of g_1 and g_2 are the same that the postcondition of g, the guard of g_1 is the conjunction of the guard of g and condition $Communication = Broadcast$ and the guard of g_2 is the conjunction of the guard of g and condition $Communication = PoitToPoint$. Moreover, the name of the

subgoal g_1 is made up from the prefix BRC and the name of g and the name of subgoal g_2 is made up from the prefix P2P and the name of g. The AtelierB tool has automatically proved the Proof Obligations associated a these OR refinements.

The designers of the SATURN network have decided to use a broadcast communication with a centralized coordination between agents and the concentrator. Thus, the following subgoals were implemented:

$$\begin{aligned}
&\textit{BRC_Centralised_Get}\ \texttt{ref_or}\ \textit{Centralised_Get} \mathrel{\widehat{=}} \\
&\quad Implementation = Centralised \wedge Communication = Broadcast \\
&\quad \rightsquigarrow s_in_l = s_in_r \qquad (13) \\
&\textit{BRC_Centralised_Put}\ \texttt{ref_or}\ \textit{Centralised_Put} \mathrel{\widehat{=}} \\
&\quad Implementation = Centralised \wedge Communication = Broadcast \\
&\quad \rightsquigarrow s_out_r = s_out_l \qquad (14)
\end{aligned}$$

In subgoals 13 and 14, it is supposed that the memory of input/output agents can be accessed directly by the concentrator, however, these memory areas are disjoint and the concentrator cannot access them directly. Therefore we need to model the communication network to specify the transfer of information between the concentrator and the input/output agents.

12.3 Refining Broadcast Centralised Get and Put

In the Broadcast Centralized Get, the concentrator requests for data to input agents and they send their remote data to the concentrator. This full duplex communication is modeled by two channels: one for request input data (req_input_ch) and another one to transfer the data ($input\_ch$). The Domain Diagram associated to this refinement provides the following typing information, indicating that the request input channel is a subset of input agents and the input channel is an association between input agents and a boolean, denoting the value of its local memory:

$$\begin{aligned}
&req_input_ch \in \mathbb{P}(MI) \\
&input_ch \in agents_in \mathrel{+\!\!\!\rightarrow} BOOL
\end{aligned}$$

The variables used to model these channels are initially set to the empty set.

The $BRC\_Centralised\_Get$ can now be MILESTONE refined using these new variables as follows:

$$\begin{aligned}
&\textit{Input_Request}\ \texttt{ref_milestone}\ \textit{BRC_Centralised_Get} \mathrel{\widehat{=}} \\
&\quad Implementation = Centralised \wedge Communication = Broadcast \\
&\quad \rightsquigarrow req_input_ch = agents_in \qquad (15) \\
&\textit{Transfer_Input_Values}\ \texttt{ref_milestone}\ \textit{BRC_Centralised_Get} \mathrel{\widehat{=}} \\
&\quad req_input_ch = agents_in \rightsquigarrow input_ch = s_in_r \qquad (16) \\
&\textit{Receive_Input_Values}\ \texttt{ref_milestone}\ \textit{BRC_Centralised_Get} \mathrel{\widehat{=}} \\
&\quad \mathrm{dom}(input_ch) = agents_in \rightsquigarrow s_in_l = input_ch \qquad (17)
\end{aligned}$$

This refinement states that three steps are need to fulfill the MILESTONE refinement. The first step, specified by subgoal 15, specifies the request for input data made by the concentrator.

The post-condition of this subgoal indicates that the request input channel must be updated with the input agents that receive the input request. The second step, specified by subgoal 16, states in its post-condition that the input channel must be updated by the input agents with their remote memory when the request input channel has been update with $agents\_in$, as indicated by the guard of this goal. Finally, the third step, specified by subgoal 17, indicates that the concentrator must update its local memory with data coming from the input channel when all input agents have put their data into the input channel.

In the Broadcast Centralized Put, the concentrator sends the computed data to output agents through an output channel $output\_ch$ having the following type in the Domain Diagram:

$$outptu\_ch \in agents\_out \mapsto BOOL$$

This variable is also initialized to the empty set. When output agents receive their respective values from the output channel, they update their memory. This behavior is specified by the MILESTONE refinement of the the $BRC\_Centralised\_Put$ goal:

$$\begin{aligned} &\textit{Transfer\_Output\_Values}\ \texttt{ref\_milestone}\ \textit{BRC\_Centralised\_Put} \mathrel{\widehat{=}} \\ &\quad Implementation = Centralised \wedge Communication = Broadcast \\ &\quad \rightsquigarrow output\_ch = s\_out\_l \qquad (18) \\ &Receive\_Output\_Values\ \texttt{ref\_milestone}\ \textit{BRC\_Centralised\_Put} \mathrel{\widehat{=}} \\ &\quad \mathrm{dom}(output\_ch) = agents\_out \rightsquigarrow s\_out\_r = output\_ch \qquad (19) \end{aligned}$$

Goal 18 specifies the sending of calculated data by the concentrator through the communication channel whereas goal 19 specifies the remote memory update made by the output agents with data coming from the output channel.

It is to be remarked that the AtelierB tool generates Proof Obligations to prove the proper execution of non-miraculous goals in a MILESTONE refinement. That is, for any goal g_i in a MILESTONE goal sequence $g_1, \ldots, g_n$, where $i \neq 1$, there is a Proof Obligation to guarantee that the guard of g_i is established by sequential execution of goals $g_1, \ldots g_{i-1}$. In this way, for example, it is proved that the guard of goal 17 ($\mathrm{dom}(input\_ch) = agents\_in$) is established by execution of goals 15 and 16. In this case, the update of $input\_ch$ with s_in_r, specified in the post-condition of 16 ($input\_ch = s\_in\_r$), ensures the guard of 17 by realizing that s_in_r is a total function with domain $agents\_in$. The AtelierB has been able to discharge automatically all the Proof Obligations in this refinement.

13 Case Study Assessment

In this Case Study we apply the Formose methodology, through the Formod and Atelier B tools, over a system already designed and built. In fact, we only explored the abstract functionality of the SATURN network without implementation details. However, this Case Study allowed us to better understand the goals formalization and the Proof Obligations associated with goal refinements.

The Domain and Goal diagrams used in sections 10 and 11 have been proved useful notions to specify, in a pictorial way, the system state and its abstract behavior. The Domain diagram provides the same information that PROPERTIES and INVARIANT clauses in a classical Event-B approach. However, this graphical view shows immediately the relationship among these elements and seems to be more suitable to non expert users. The Goal diagram becomes

a declarative specification of the system behavior at any abstraction level. The intended role of these diagrams is similar to the role of events in a Event-B formalization. Nevertheless, the computation model of an Event-B system, that is, iteration over events chosen in a non deterministic way, adds complexity in the ordering of actions to be done by the system, which is avoided by the refinement of Goals proposed by the Formose methodology. Moreover, new events in Event-B are difficult to be introduced and need to declare variant expressions to prove bounded stuttering steps; the Formose approach also avoids this complexity by introducing new goals in a natural way through its three kind of refinements.

Refinements of goals have been rigorously proved thanks to the Proof Obligations defined for the three kinds of goal refinements. This fact was illustrated in section 11 where the *Saturn* goal was first defined as an AND refinement of subgoals *Get* and *Control*, then after the impossibility of discharge the Proof Obligations of this kind of refinement, we defined it as a MILESTONE refinement. Moreover, in the paragraph “Refining Broadcast Centralised Get and Put” in section 12.3, it was shown how the Proof Obligations guarantee the correct sequencing of goals in a MILESTONE refinement.

From our experience in the application of the methods and tools of the Formose methodology, we have two comments to make. The first one concerns the methodological point of view of Goal Refinements, where the refinement constructs of the Formose methodology have been proved useful at the abstract level where they were applied in this Case Study. Nevertheless, a new construct to express iteration of goals seems to be necessary if algorithmic details are to be specified. The second comment concerns the Atelier B integration in the Formod tool. As indicated in section 3.3.2 "Proof Activity and Feedback to Formod" of deliverable D3.2.b/D3.2.c, the integration of the Atelier B in the Formod tool needs to be improved, in order to get a traceability between the Proof Obligations of the Atelier B and the refinements in the Goal Diagrams.

However, the current state of methods and tools proposed by the Formose methodology are showed to be useful in the field of Requirements Engineering by providing a formal relation, made up of Goal and Domain Diagrams, between requirements in natural language and their respective formalization in the Event B notation.

14 Scope

14.1 Identification

Document Name:	Assessment report on the Formose method and tools on the use case studies
Deliverable Identification Number:	D1.2
Project Name:	ANR Formose
File Name:	Formose-D1.2