# Formal architecture modeling for documenting and assessing Aeronautics Maintenance: A case study

Olivier Poitou and Pierre Bieber
ONERA, Département Traitement de l'Information et Modélisation
2, avenue Edouard Belin BP74025, 31055 TOULOUSE Cedex 4
firstname.lastname@onera.fr
Joël Ferreira, TAP Portugal, jtferreira@tap.pt
Ludovic Simon, Thales Avionics, ludovic.simon@fr.thalesgroup.com

JANUARY 31st to FEBRUARY 2nd 2018

## Abstract

This paper presents a case study that will illustrate what a dedicated formal specification can bring to a socio-technical system description and how it can take place. The illustration will use an ONERA specification language and tool and apply it on the domain of aeronautical maintenance that is being dealt with in the Clean Sky 2 European project ADVANCE-AIRMES. As opposed to informal modeling, formal specification will offer: (1) to automatically assess some high level properties or indicators and hence compare candidate solutions, (2) to produce a standard documentation conforming for example to the NATO Architecture Framework (NAF), (3) to better support safety and security analysis.

***Keywords :*** formal specification, free modeling, socio-technical systems, systems of systems, aeronautical maintenance, safety assessment, security assessment

## 1 Introduction

This paper introduces a case study from the AIRMES European project about Aeronautical maintenance. It describes how different modeling and analysis processes of this socio-technical system took place: structural and behavioural formal architecture modeling, high level performance indicators computation, safety and security assessment support. The following sections reflect this order, section 2 introduces the AIRMES project: section 3 is dedicated to the way formal modeling of the whole system took place starting from heterogeneous input,4 introduces the standard documentation production from the obtained model. Section 5 introduces some indicators computation and sections 6 and 7 explains how those analysis have been supported. Then section 8 sums up some lessons learnt, introduces some perspectives and concludes this paper.

## 2 Context

The ADVANCE-AIRMES[1], AIRMES standing for Airline Maintenance Operations implementation of an E2E Maintenance Service Architecture and its enablers, is a H2020 CleanSky2 European Project. It focusses on optimising end-to-end aeronautical maintenance activities within an operator's environment. Several participants bring some innovation to, so-called, "technical enablers" such as: an integrated health monitoring and management (IHMM) platform, contextualized documentation, prognostics or structure health monitoring, but also a Virtual Reality technology for improving documentation and training of the maintenance technician or Augmented Reality support to the maintenance technician in operation. Hence, the integration of those technologies into a whole end to end (lately abreviated in E2E) system is really a key to fulfil the project high level objectives. Among the project partners[2], those contributing to this paper are listed below. TAP company, the Portuguese flag carrier, is both the coordinator as well as the aircraft maintenance business expert. THALES is a core partner having in charge, in particular, a critical component of the system: the IHMM platform, the E2E security assessment as well as a coordination function and a high business expertise. ONERA is in charge of modeling the global system architecture for documentation purposes as well as for evaluation purposes. This evaluation is multiple, it encompasses computing

[1]This project has received funding from the Clean Sky 2 Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 681858, AIRMES project, http://www.airmes-project.eu project

[2]Full list of AIRMES partners is: AIS, ATOS Origin, Cranfield University, ISQ, L-UP, MEGGIT, M2P Consulting, ONERA, PTC, TAP, Technische Universiteit Delft, Tekever. Advance Level Core Partners also involved in AIRMES are Thales and Airbus

some correctness features as well as some performance aspects and to support safety and security assessments. ONERA is also in charge of the safety evaluation.

## 3 Gathering the specification

At the first stages of the project, E2E system specification/documentation encompassed a lot of documents, each of them representing a piece of information about the system. Each of those documents uses its own description language and adresses one or several concerns, inside a given perimeter at a given level of detail. At this moment, it is difficult to say exactly what is *the* documentation of the whole E2E system, and hence impossible to demonstrate that this documentation is complete and consistent.

From this point, the approach has been to gather all those pieces of information about the system, to identify their commonalities and eventually to have a way to "query" and reason about the resulting information base.

Querying and reasoning about the information means that this information must be formally modeled. Relying on a Domain Specific formal Modeling Language (DSML) has become a standard in such situations. The most current approach is then to first fully define the syntaxes and semantic of the DSML and then to describe the architecture itself using the defined DSML. However this approach does not address the issue of the heterogeneity of the inputs, unless defining a very complex DSML which can be hard to manage if monolytic.

The approach followed during this project differs from the most standard ones by two ways:

- based on the available inputs, a co-construction process has been used to build at the same time both the model of the E2E system and its metamodel. This approach has some strong similarities with the so called free-modeling approach described in the following work [3].
- instead of trying to build a unique DSML, a metamodel for each different input has been built. The commonalities being managed by using reference to the first model or meta-model introducing the corresponding item(s).

To support this approach, we relied on Weird, an ONERA specification language for expert modeler (anterior version of the language has been introduced in [1]). Weird provides some fundamental constructions to define concepts, entities, relations between model elements (with the special case of applications) and functions (between model elements and primary types as integer or boolean) as well as a rather rich expression language that will be used to describe some model

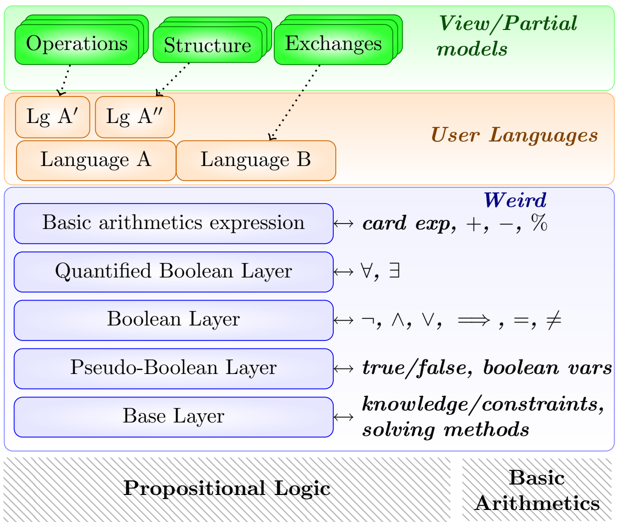


**Fig. 1.** Architecture of the modeling tooling

elements as well as to query the model for assessments or simple observations. Weird typing is very flexible (encompassing subconcepts, multiply typed or untyped entities), in fact typing is considered (internally) as a relation special case between entities and concepts (some would say a metarelation). Weird also offers a modularity mechanism, by the *world* and *derives* keywords. When focusing on meta-modeling, this modularity will be used to separate the different DSML as well as to differentiate some abstraction levels inside them. For now Weird semantics only have a pragmatic definition (ONERA prototype being the reference implementation) but a denotational or at least translational one could be produced in terms of propositional logic. Weird input concrete syntax is textual, then it has two concrete rendering syntaxes a textual one (the same as the input syntax) and a graphical one. The graphical rendering syntax is customizable, offering the modeler to define full domain specific languages and not only a dedicated meta-model [5].

Resulting Weird tool architecture and usage overview is presented in figure 1. In this figure, the proposition logic and basic arithmetics roots are recalled, as well as the different language enrichment layers that constitute the Weird language. Illustration of its usage is also represented with the example of two user languages $A$ and $B$ built upon Weird as well as two languages A′ and A″ built upon $A$. On the top of figure 1 , user partial models/views are also represented, several views being associated with each corresponding language. What does not appear in this figure is the fact that each user language may be constituted of two parts one mandatory describing the metamodel from which a textual concrete syntax will be directly cascaded; and one optional offering to customize the graphical rendering of the diagrams associated to the language.

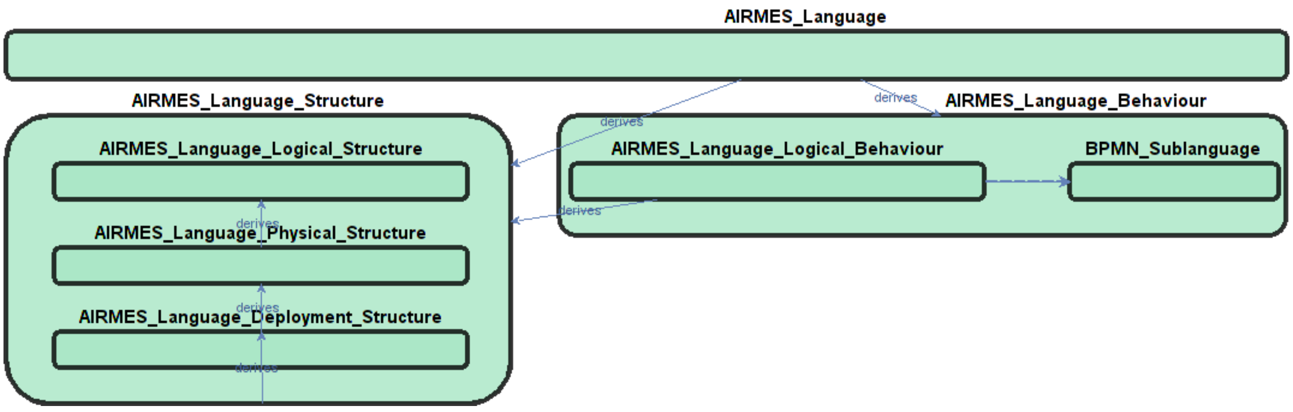
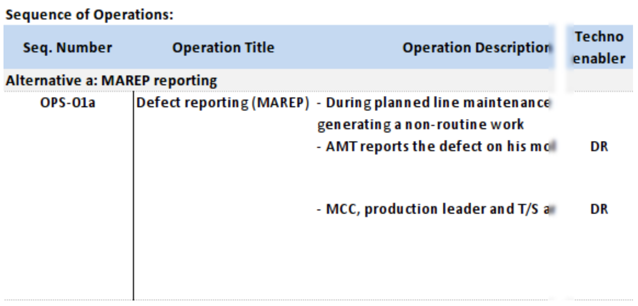

AIRMES_Language
AIRMES_Language_Structure
AIRMES_Language_Logical_Structure
AIRMES_Language_Physical_Structure
AIRMES_Language_Deployment_Structure
AIRMES_Language_Behaviour
AIRMES_Language_Logical_Behaviour
BPMN_Sublanguage
derives

**Fig. 2.** Graphical view of the links between the user languages that were produced or used in AIRMES


Sequence of Operations:

| Seq. Number | Operation Title | Operation Description | Techno enabler |
|---|---|---|---|
| Alternative a: MAREP reporting | | | |
| OPS-01a | Defect reporting (MAREP) | - During planned line maintenance generating a non-routine work | |
| | | - AMT reports the defect on his mo | DR |
| | | - MCC, production leader and T/S a | DR |

**Fig. 3.** Operational scenario extract in Excel

Figure 2 presents the AIRMES produced languages where three structural languages representing three abstraction levels were used, as well as a behavioural language, itself based on a BPMN-inspired language (BPMN standing for Business Process Model and Notation[7]).

To build a standard architecture description of the E2E system studied in AIRMES project, relevant documents were chosen. They had different natures (different diagrams with different notations, full-text documents, tables) and were describing different aspects of the system: behavioural as well as structural. Because written by business experts with a high practical knowledge, this input mainly focus on physical level items or even on deployed level items (as opposed to logical ones). But anyway, those input elements were first took as they were: a mix of information from those different (and not yet formalized) abstraction levels.

For instance, behaviour description had the form of Microsoft Excel tables (see Figure 3) describing operational scenarios as the most representative sequences of operations taking place to deal with the major maintenance situation an MRO (Maintenance, Repair and Overhaul) has to face. Some scenario items were decorated with rationale formulated as business rule (that would have taken place in a logical level description). On the opposite, some indications on equipment or human roles were sometimes refering to deployed elements.

Applying the co-construction process, as much raw

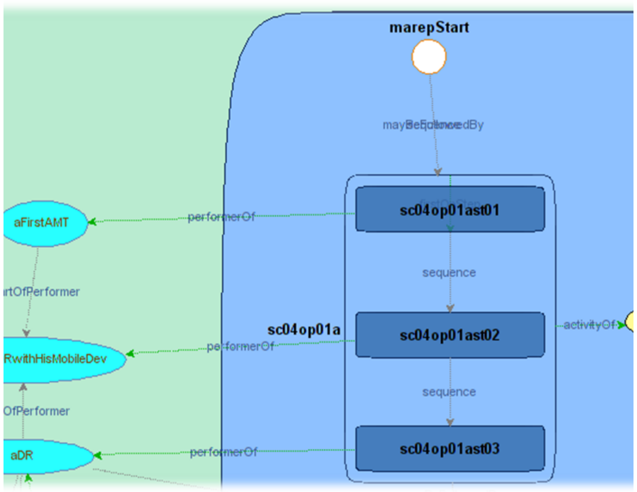


**Fig. 4.** Graphical view of the Weird code describing the same operation as figure 3

information as possible was extracted from every input to the co-constructed formal specification languages and models, as if building a knowledge database (and its scheme at the same time) (see Figures 4 and 5).

Concretely every input has first been considered separately, producing a single model file. Each input in a set of inputs having the same document nature was using and contributing to the same user langage definition.

The second step has been to integrate those models and languages. Here the tool was helping by pointing at redundancy in entities, concepts or relation definition. Use of some known business rules or some logically induced and abducted knowledge offered us to both check the obtained model consistency as well as to complete the model where possible. Of course, discussions with experts complemented this investigation to correct or complete the resulting model.

The result of this first phase is a consistent model and language set (see figure 2) , i.e. no knowledge item couple is contradictory. Completeness is not reached but in such contexts as systems of systems and socio-technical systems this should not be considered as a

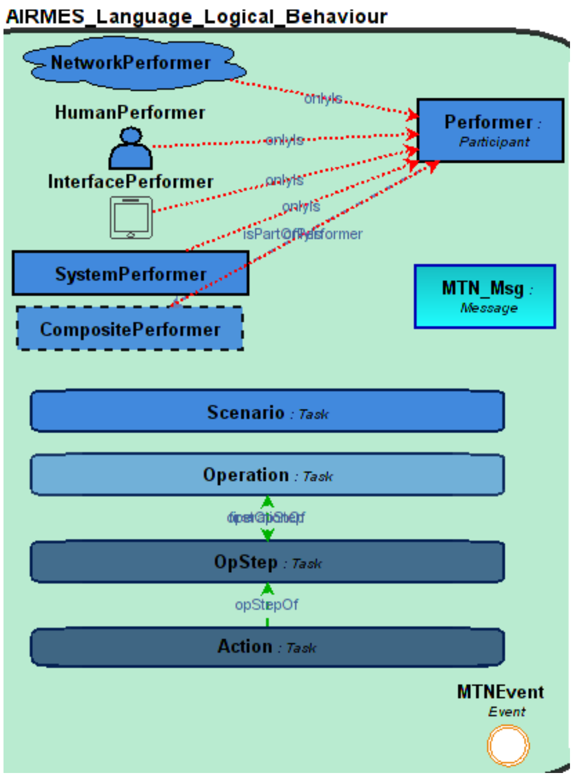


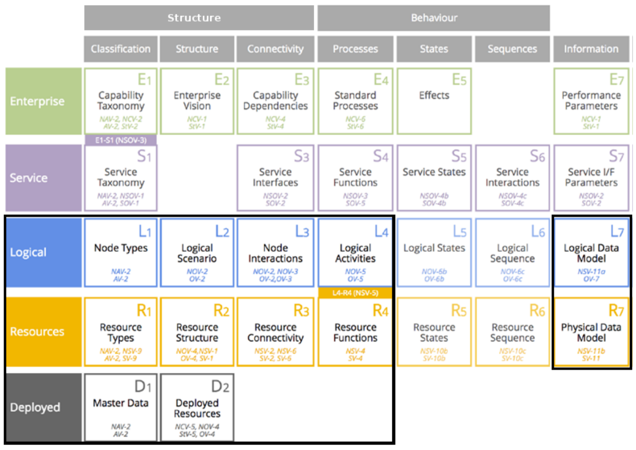
**Fig. 5.** Graphical view of the behaviour metamodel produced to support AIRMES operational scenarios (deriving from a classical BPMN, not shown here)

blocking property since this is a consequence of the nature of those systems by itself (no stakeholder has a complete full-detailed view of the whole and some elements are not fully known).

Note that, at this point, the model contains too much information to offer a suitable graphical view that would offer a human the opportunity to efficiently obtain information on the system.

The second phase is then to exploit this model. This may take place in multiple directions and is supported by WEIRD prototype with what could be called *query based diagramming*. The idea is to build new diagrams/views of the same model following some user queries. Those queries may be very basic as a concept breakdown into sub-concepts and related entities (leading to a kind of taxonomy diagram), a relation instanciation diagram (showing all entity linked by the selected relation) or an entity focused diagram (all knowledge about a given entity). Each of those basic queries may be adjusted afterwards by some items deletion or addition to build some more useful diagrams.

A first exploitation of the model via this technique is the discussion with experts based on some focused extractions of the model to check and improve its correctness with regards to the reality (one could talk about soundness). This direction will not be developped any further in this paper.



| | Structure | | | Behaviour | | | |
|---|---|---|---|---|---|---|---|
| | Classification | Structure | Connectivity | Processes | States | Sequences | Information |
| Enterprise | E1 Capability Taxonomy NAV-2, NCV-2 AV-2, StV-2 | E2 Enterprise Vision NCV-1 StV-1 | E3 Capability Dependencies NCV-4 StV-4 | E4 Standard Processes NCV-6 StV-6 | E5 Effects | | E7 Performance Parameters NCV-1 StV-1 |
| | E1-S1 (NSOV-3) | | | | | | |
| Service | S1 Service Taxonomy NAV-2, NSOV-1 AV-2, SOV-1 | | S3 Service Interfaces NSOV-2 SOV-2 | S4 Service Functions NSOV-3 SOV-5 | S5 Service States NSOV-4b SOV-4b | S6 Service Interactions NSOV-4c SOV-4c | S7 Service I/F Parameters NSOV-2 SOV-2 |
| Logical | L1 Node Types NAV-2 AV-2 | L2 Logical Scenario NOV-2 OV-2 | L3 Node Interactions NOV-2, NOV-3 OV-2,OV-3 | L4 Logical Activities NOV-5 OV-5 | L5 Logical States NOV-6b OV-6b | L6 Logical Sequence NOV-6c OV-6c | L7 Logical Data Model NSV-11a OV-7 |
| | | | | L4-R4 (NSV-5) | | | |
| Resources | R1 Resource Types NAV-2, NSV-9 AV-2, SV-9 | R2 Resource Structure NOV-4,NSV-1 OV-4, SV-1 | R3 Resource Connectivity NSV-2, NSV-6 SV-2, SV-6 | R4 Resource Functions NSV-4 SV-4 | R5 Resource States NSV-10b SV-10b | R6 Resource Sequence NSV-10c SV-10c | R7 Physical Data Model NSV-11b SV-11 |
| Deployed | D1 Master Data NAV-2 AV-2 | D2 Deployed Resources NCV-5, NOV-4 StV-5, OV-4 | | | | | |

**Fig. 6.** NAF v4 viewpoints overview grid (partial) with AIRMES relevant views emphasized

Another direction is to automatically produce an architecture documentation of the system that conforms to a standard as the NAF.

## 4 Standardizing the architecture documentation

Let's first clarify some definitions, the *architecture* we were aiming to document is defined in the ISO/IEC 42010[4] as:

> *Fundamental concepts or properties of a system [enterprise] in its environment embodied in its elements, relationships, and in the principles of its design and evolution*

Following the same standard, the architecture description is made of *architecture views*, defined as:

> *Work product expressing the architecture of a system from the perspective of specific system concerns.*

As a consequence, it is important to understand that there is not a single "architecture diagram" (do not look for it in this paper) but, on the opposite, that the architecture documentation is a set of architecture views documenting system concerns via a set of diagrams (some of which are introduced in the coming chapters).

In this study, selection of the system concerns was based on available specification documents. Organization of those concerns and guidance for producing the relevant views rely on an architecture framework: the NAF[6].

By organizing the documentation in standard viewpoints and associated views, NAF offers to explicit the intent of each documentation element (see [6]). NAF introduces a lot of standard views, organized as a grid

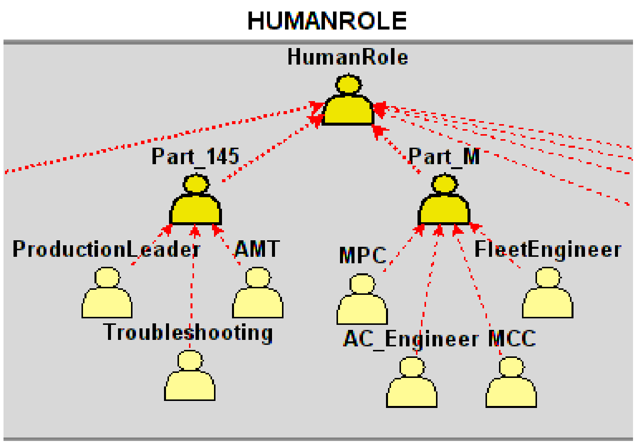


**Fig. 7.** Example of produced NAF-R1 diagram: subtypes and instances of human roles introduced by different documents are here organized as a typology diagram. Lacking or redundant roles are then more easily identified by business experts.

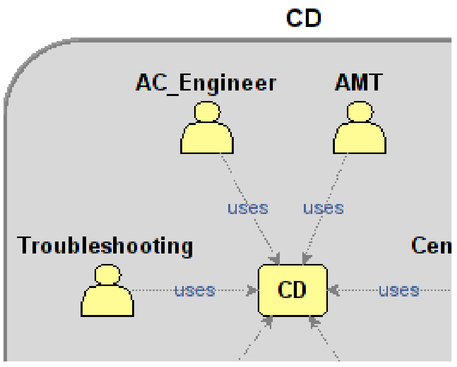


**Fig. 8.** Example of produced NAF-R2 diagram: identification of all the users of a system

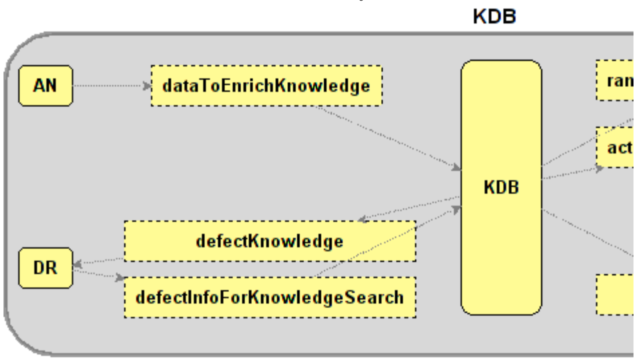


**Fig. 9.** Example of produced NAF-R2 diagram: data exchanges of a system, here a knowledge base, with other systems.

with roughly the concern nature as the horizontal dimension and the abstraction level as the vertical dimension (see figure 6). Only the most relevant views to AIRMES were retained for the study.

Then the way they may be extracted from the whole model have been defined to be able to produce them automatically using the query based diagramming ability of WEIRD.

For instance, the first column of the NAFv4 grid focuses on providing taxonomies of the items appearing in the architecture description. This can be produced, for any concept of the model chosen as root, by building the tree of all its subconcepts and all instances of each of those concepts. Depending of the chosen concept as root, NAF-L1 (logical level), NAF-R1 (physical/resources level) or NAF-D1 (deployed level) can be produced for documentation purposes. Those views being updatable at any moment from the actual version of the model (see Figure 7 for a partial example). Other abstraction layers had not been developped during the project but the mechanism would have been the same. Others "columns" of the NAF are considered (see figures 8 and 9 for examples), and even for those that were not selected, the principles would, again, have been the same.

Having put in place this documentation production chain then ensures that standard or customized diagrams may be automatically updated when any knowledge is modified, added to or removed from the system model.

In fact, producing NAF compliant diagrams is really just a simple application of the query based diagramming mechanism. More generally, this mechanism can be seen as a relevance filter of a set of knowledges towards the information need of a user. The idea is to extract information that matches a user profile (role played, skill/interest) and/or an information request. For example "a network expert seeking information of connectivity of the equipment E" or a "business stakeholder asking for connectivity inside the system" will correspond to different formal requests (each of them corresponding to columns 3 of NAF-v4 grid) that will be done to the model to first select relevant knowledge and then organize it into a diagram (in our prototype tool that diagram may then be adjusted: layout change, items removal, ...). Note that inputs are often very technical, and to give someone a high level view, extracting information is not as simple as keeping or dropping knowledge items but may require to "abstract" some of them. A typical example would obviously occur in the second request of the above example: any wires, hubs or other technical communication equipments will not be considered as relevant because of the high level view a business stakeholder is expecting; but the path that these communication equipments create between two relevant equipments or platforms then has to be "abstracted" to fulfill the connectivity aspect of the request.

Readability is also a key in the production of the documentation, and for this purpose each produced diagram should contain a limited number of elements. As a consequence the whole documentation is made of an

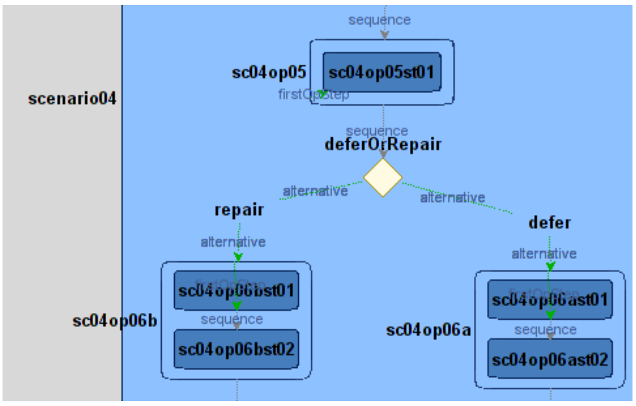


**Fig. 10.** Scenario extract: Operations contain operational steps, alternatives link a decision point to an operation depending on some observable

important number of diagrams. That point leads to the perhaps trivial but not yet fully accepted remark that producing paper-based snapshots of the architecture diagrams may not be the most effective way to document an architecture as compared to interactive exploration with a tool.

## 5 Evaluating indicators

Besides some boolean conformity checks, some other indicators may be computable early in the architecture definition process. Of course, they are mainly high level ones but anyway they are useful to support decision maker by comparing several candidate solution scores with respect to these indicators. One example is the worst scenario path durations.

Operational scenarios may contain several starting events, several ending states, and, between them, different sequences of operations, of choices, of joins may take place. In our case study, scenario are sets of linked operations. Each operation is a sequence of Operational Step, and each operational step may contain several actions (that can take place in any order and/or concurrently). Finnest grain, Operational Step or Action, is supported by function(s) allocated to so called "Enablers". Technology providers are able to provide worst case execution times of all the functions allocated on the enabler they are in charge. A rough worst case duration time can then be calculated from the preceeding knowledge[3] (see listing 1):

- each action is given a cost based on the sum of each of its supporting function costs, sum of access times to each enabler involved as well as an estimation of the human working effort associated to the operation (acquisition of the computation results and/or human actions to realize consequently),

[3]the following description is not the exact way the computation occurs but illustrate the idea

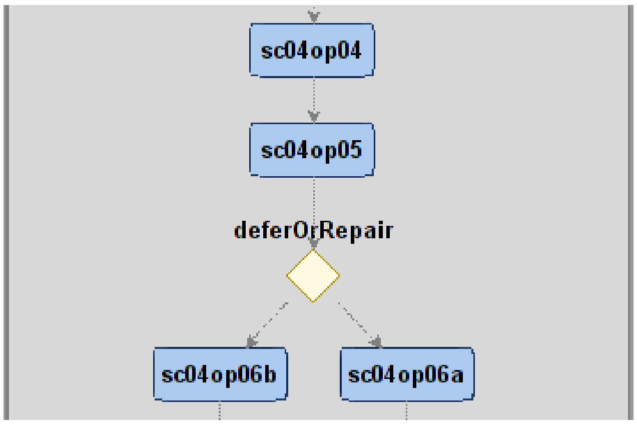


**Fig. 11.** Scenario path abstraction: Operational steps are ignored, observable of alternatives is ignored to link all elements with the abstract relation *mayBeFollowedBy*. The decision point by itself is kept because it may convey some associated cost.

  this cost is then added to the costs corresponding to the same computation on actions that are grouped under the same operation step to produce the operational step level cost.

- then operational step costs are sumed up into operation level costs
- then every scenario path cost is computed by summing the operation cost of any operation being on the path

Note that a scenario path is not a notion that is readily available, since relation between the operations are not always a sequence relation but may also be a decision point–alternative couple or a join point (see 10).

An abstraction of those different kinds of relations is hence first done into a single more abstract relation that offers to easily compute paths (see 11).

To achieve this the facts system of our prototype has been used: a fact is an expression of the form: *for (typed) variables such that "selection expression" then add "knowledge" to the model.* This system is the same that is used to encode business rules that complete the model without requiring that the user explicitly provides some information. Even if it may not be the best way to do so, a simple example is a fact encoding that a binary relation $r$ is symetric by adding the symetrical case every time a couple is inserted in the relation $r$ by the user. That would be encoded `fact any (e1,e2) in r | true ==> (e2,e1) in r`.

Listing 1: Listing of the cost computation expressed in Weird (partial)

```
/* compute the cost associated to a
   function inside an operation */
fact any entities o:Operation, f::Function
    | (o,f) in supportingFunction
  ==> costByFunction[o,f] =
            analysisTimeOf(f) +
            processTimeOf(f,hostedBy(f))+
            accessTimeOf(hostedBy(f))

/* compute the cost at operation level
   based on function by function costs */
fact any entities o:Operation | true
   ==> cost[o] =
        sum of
            costByFunction(o,f::Function) |
            ((o,f) in supportingFunction)

/* Compute the total Cost of every path
   from a start point to an end one */
fact any entities s:StartEvent,e:EndEvent
  | true
  ==> totalCost[s,e] =
        sum of cost[o:Operation]
            | ((s,o) in ^mayBeFollowedBy and
               (o,e) in ^mayBeFollowedBy)
```

In addition to numeric indicators measuring for example times, costs or number of items having some properties, boolean and enumeration indicators may also be computed. Here are some examples of indicators envisaged or dealt with in the project: all functions are allocated to at least one equipment (boolean version), list of functions that are not allocated to any equipment (enumeration value), list of moves between location a technician has to make during a given scenario, "undoable" operations (operation for which at least one supporting function is either not allocated or allocated to an equipment (or a human) that can not execute it (having a role to which the function can not be assigned – by lack of skill or accreditation for instance)

## 6 Safety assessment support

The approach followed in this study is close to the one described in [2]. Our first plan concerning improvement of safety assessment support from the architecture model lied in enforcing the link between the architecture model and the one AltaRica uses for its analysis. More exactly to make the Altarica input format a rendering format of our prototype.

After a further observation of the system behaviour descriptions we had, we realised that, at the level of abstraction we were working, the AIRMES system under study did not encompass any safety mecanism like redudancy or voting. In fact only some operations could fail and some others, involving checks, could act as barrier to fault propagation. This specific context leads us to change our plan and to deal with the fault propagation computation directly in our prototype without relying on AltaRica.

To put it simple, the safety assessment relevant to this context needs three inputs: the operational paths, identified operations that may fail or underperform and identified operations that would detect and/or correct a fault propagation [4].

Operational paths are in fact abstraction of different links between operation, the *sequence* between the operations of course, but also the choices/alternatives and join/merge operations. Those paths are the same as the ones computed for the cost indicator computation (see 5 for a full description).

Depending on their nature, operation may introduce fault, or be considered as neutral (propagating faults without introducing new ones nor modifying existing ones), or they may be barrier (detecting or correcting faults).

To define the safety input language, in addition to the three notions just mentioned, the concept of *FailureMode* was also added along with four associated entities: *ok,miss, lta* (less than adequate) and *opposite.* Those failure modes are associated to operations via a function *hasFailureMode* indicating when an operation may introduce a fault of this nature. They may also be associated with operation by the *mitigates* application indicating that an operation will modify a given fault info a different nature of fault (if the latter is *ok*, then the operation is fully correcting the fault but it may also only detect the fault and then a *lta* becomes a *miss* for example).

The idea then is to check that, for any operation that may become faulty, the fault can not propagate to a critical operation (one for which the fault would not be acceptable) – said differently to check that there is a barrier (an operation that at least detects the fault or even better repairs the fault) on the path between any possibly faulty operation and a critical operation. In WEIRD, it has been possible for us to write an expression building the list of all operations that may lead to a given operation and failure mode couple (see listing 2). This occurs in two cases, either an operation may introduce the unwanted failure and no barrier exists between the operation and the observed end point, or an operation introduces a different (worse) failure that a barrier on the way reduces to the –still undesirable– failure that reaches the end point (without being itself reduced again by another barrier). Note that the expression in listing 2 does not take into account a "reduction" of the fault in more than two steps. In practice we could find no example of such reduction and we strongly believe that this is representative enough of a lot of contexts, taking into account the little number of fault "levels" in presence. For example a first barrier may be able to reduce both *less than adequate* and *opposite* levels to a *miss* (detection barrier) and then a second barrier may reduce a *miss* to *ok* (correction barrier).

[4]In the context of this study, those subsets of operations are disjoints: barriers are safe and never fail

Listing 2: Building the list of operation that may cause a miss at end of scenario 09

```
  observe EndwithAMiss=
     entities o:UnsafeOperation |
      (o,sc09End) in ^mayBeFollowedBy and (
        (hasFailureMode(o,miss) and
          not(exists entity b:Barrier|
            isDefined(mitigates(b,miss)) and
         ((o,b) in ^mayBeFollowedBy) and
         ((b,sc09End) in ^mayBeFollowedBy)))
       or
        (exists entity f:FailureMode |
              hasFailureMode(o,f) and
          (exists entity b:Barrier|
                  mitigates(b,f)==miss and
          ((o,b) in ^mayBeFollowedBy) and
          ((b,sc09End) in ^mayBeFollowedBy) and
          not(exists entity c:Barrier|
             isDefined(mitigates(c,miss)) and
          ((b,c) in ^mayBeFollowedBy and
          ((c,sc09End) in ^mayBeFollowedBy)))))
       )
```

## 7 Security assessment support

A similar approach was initiated towards supporting security assessment. Security experts identified some primary assets: data with an associated security expectation as availability, integrity or confidentiality. The concept of PrimaryAsset has hence been added to a security language inheriting from the architecture language. In addition, three applications have been introduced in the language to link data to security expectations: (*isAvailabilitySensible*, *isIntegritySensible* and *isConfidential*).

With those elements, knowledge is added to the architecture model to describe the security expectations with regards to data.

Then a supporting assets identification process may be used to:

- enumerate the supporting assets of a given primary asset or
- to identify the security expectations (and hence requirements) a given technology enabler has to fulfil depending upon the primary asset it, or one of its parts, may manipulate (considering or not simple data transit via the enabler).

Listing 3: Building the list of supporting assets from the primary ones

```
// fill-in a manipulateData relation
fact any entities (e1,e2,d) in communicatesData
     | true ==> (e1,d) in manipulateData
fact any entities (e1,e2,d) in communicatesData
     | true ==> (e2,d) in manipulateData

// Append to any data having a security
// expectation the PrimaryAsset type
fact any entities d:Data |
   isConfidentialitySensible(d) or
   isAvailabilitySensible(d) or
   isIntegritySensible(d) ==> d:PrimaryAsset

// use the previous elements to fill-in
// the isSupportingAsset relation that identifies
// all primary to supporting assets links
fact any entities e:Enabler,p:PrimaryAsset
       | (e,p) in manipulateData
         ==> (e,p) in isSupportingAsset

// To be able to build an exploitable table
// of primary -> supporting assets
fact any entities (e,p) in isSupportingAsset
     | true ==>   (p,list,e) in supportingAssetList
```

In listing 3, a relation is built that links primary assets and their supporting assets. This relation is then available to build more precise requests like the ones suggested in the previous paragraphs. It is also used to build a slightly reworked version of itself (last lines of the listing) *supportingAssetList* from which our prototype is able to produce a CSV file. That file may then be opened with an Excel-like tool, for instance to continue the analysis in a dedicated Excel Template based process (as it was the case in the AIRMES project).

Of course the security analysis is far from being completed with only those elements, but we did not have the opportunity to experiment further in the context of this specific project.

## 8 Conclusion and perspectives

In this paper, a case study of modeling a complex socio-technical system was introduced. System complexity leads to distribute its specification among several contributors with hers own skills, focus and languages. In such context, consistency and completeness are no more ensured and we believe that formal modeling is the best way to keep this deviation under control. Choosing any formal language and using it to (try to) model all aspects of all the system is no more an option in this context considering the size of the final model and the variety of concerns to be addressed. On the opposite we propose to use a minimal formal language to build both submodels of the system and their accompanying dedicated languages – as close as possible to the ones chosen by the specification contributors (or to develop import/export tools between the contributor favorite language and the formal one).

We also propose to use standard views promoted in framework like the NAF as rendering views and not automatically as input views. If a viewpoint definition may guide one to produce its specification contribution, it should never be considered as a strict limitation. Specification "pieces" with unavoidable overlapping are a valuable input as soon as formally processed. It helps in better identifying articulations between the specification pieces, discussions to have when inconsistency appears and finally bring a better system model. In addition, as soon as the modeling is formal, then rendering standard documentation becomes possible from the whole set of inputs. From the experiment we led in the AIRMES project, a lesson learnt is that the level of maturity required on specification inputs is hard to define. With most partners, less architecture modeling aware, the process should have started earlier that what we have done, to guide them and devise the language and the information they may provide and give them feedback on a very regular basis. With more experienced ones (from the architecture modeling point of view), early integration may not be the best option since they may "jump" from languages to languages to model and discuss internally. Trying to reflect those internal steps in the global process is time consuming without showing a strong added value.

This work also strenghten our position that an agile modeling approach is the most appropriate in that kind of context. Making the language formality requirement and the agility requirement compatible needs that the tooling evolves and integrates a smarter support to the architects. One issue lies in the verbosity of formal languages that makes it difficult to fully describe a large system manually. Business rules, domain rules, project rules... should be encoded and exploited to better support the architect modeling work. For example, knowing constraints put on an allocation relation may be exploited -by an underlying constraints solver- to suggest a valid one to the architect during the modeling process and to quickly react and adjust the proposition as the architect continues to manually add knowledge until she accept the tool proposition. Typing of elements in the architecture may also be computed inspired from what is done in some programming languages. Inputs merging is another key point in modeling a complex system, exploiting knowledge on the intent of the input and/or the contributor itself may offer a tool to better support the architect in its merging decision making. For example, some reliability estimation on given topics/elements may be associated to sources of information and then taken into account in the merging process like in the information fusion research domain. Detecting items that are given different names in different inputs but are in fact one single item, or, on the opposite, a same name being used to designate to different things are issues that have some solutions from research on ontologies alignment that may benefit in the context of architecture modeling. A constraint solver may also be used in order to deal with lack of completeness of the model. Indeed, incompleteness is almost inherent to system of systems modeling where detailed information about some elements may not be shared to every partner by the owner of the elements, and/or where the dynamicity of the whole system bring some incertitude upon the exact configuration in place at a given evaluation time. In those cases, evaluation could be made upon possible values computed by a constraint solver where information is missing, leading to being able to compute values (even if they are approximation, maximisation, ranges...) where no computation at all could occur otherwise. It can also support modeling by informing the architect modeler when the model can no longer satisfy some rules whatever the complement of the model is and/or suggests some modeling action (like the introduction of some additional entities). All of these perspectives are currently envisaged, in addition to some syntax improvement of our prototype tool.

By the effort that had to be paid by manually doing some clearly automatable tasks, the AIRMES project emphasized the work that still needs to be done in improving the modeling process support by a tool. But it also conforted us in the position that formal modeling is required to detect consistency issues and lack of information and consequently guide discussions and specification work in a large project, to eventually get a highly valuable (formal) system model from which to obtain accurate and reliable documentation and evaluations.

## References

[1] Pierre Bieber et al. "MIMOSA: Towards a model driven certification process". In: *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*. TOULOUSE, France, Jan. 2016. URL: `https://hal.archives-ouvertes.fr/hal-01289704`.

[2] Pierre Bieber et al. "Model Based Safety Assessment of Concept of Operations for Drones". In: *2E-Etudes probabilistes de sûreté 2* (2016).

[3] Fahad R Golra et al. "Using free modeling as an Agile method for developing domain specific modeling languages". In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. ACM. 2016, pp. 24–34.

[4] ISO/IEC/(IEEE). *ISO/IEC 42010 (IEEE Std) 1471-2000 : Systems and Software engineering - Recomended practice for architectural description of software-intensive systems.* July 2007.

[5] Anneke Kleppe. "A Language Description is More than a Metamodel". In: *Fourth International Workshop on Software Language Engineering*. Grenoble, France: megaplanet.org, Oct. 2007.

[6] NATO. *NATO Architecture Framework version 4.* URL: `http://nafdocs.org/viewpoints/` (visited on 2017).

[7] Object Management Group. *Business Process Model and Notation.* URL: `http://www.bpmn.org/` (visited on 2017).