Synchronization of System Architecture and Safety Models: a Proof of Concept

Michel Batteux, Jean-Yves Choley, Faida Mhenni, Tatiana Prosvirnova, Antoine Rauzy

Synchronization of System Architecture and Safety Models: a Proof of Concept

Michel Batteux *, Jean-Yves Choley †, Faïda Mhenni †, Tatiana Prosvirnova ‡, Antoine Rauzy §
* *IRT SystemX*, Palaiseau, France
Email: michel.batteux@irt-systemx.fr
† *Quartz Laboratoire*, *Supmeca*, Saint-Ouen, France
Email: jean-yves.choley@supmeca.fr, faida.mhenni@supmeca.fr
‡ *ONERA/DTIS*, *UFTMiP*, Toulouse, France
Email: tatiana.prosvirnova@onera.fr
§ *dept. of Mechanical and Industrial Engineering*, *Norwegian University of Science and Technology*,
Trondheim, Norway, Email: antoine.rauzy@ntnu.no

***Abstract*—To face the increasing complexity of technical systems, engineers are designing models. The integration of models coming from various engineering disciplines, such as system architecture, multi-physics simulation, automatic code generation as well as safety and performance analyses, is one of today's industrial challenges.**

In this article we present model synchronization – a framework to ensure consistency between models coming from different engineering domains, based on S2ML (System Structure Modeling Language). We illustrate our purpose using an Electro-Mechanical Actuator (EMA) of an aileron for a small aircraft. We show how the introduced framework can be used to handle consistency between system architecture models (represented in SysML) and safety models (represented in AltaRica 3.0) with several architecture variants.

***Index Terms*—Model synchronization, model structuring, SysML, AltaRica, S2ML**

I. Introduction

Technical systems are getting more and more complex. To face the increasing complexity of systems, engineers are designing models. These models have different maturity, are designed at different abstraction levels and for different purposes. The integration of models coming from various engineering disciplines, such as system architecture, control, multi-physics simulation, automatic code generation, safety and performances analyses, is one of today's industrial challenges.

Collaborative data bases (PDM/PLM) and tools to set up traceability links between models provide a support to manage models in version and configuration, but not to ensure consistency between them. Model transformation techniques [4], [10], [15] assume a master/slaves organization of models, which is not realistic in practice.

In this article we present model synchronization – a framework to ensure consistency between models coming from different engineering domains.

This framework is based on the thesis that systems engineering modeling formalisms are made of two parts:

- An underlying mathematical framework, which aims at capturing some aspects of the system behavior, e.g. differential equations for Modelica [6] and Matlab-Simulink [8], Data-Flow equations for Lustre [7], Guarded Transition Systems for AltaRica 3.0;
- A structuring paradigm that makes it possible to build and organize models by assembling parts into hierarchical descriptions.

Behavioral descriptions are specific to each engineering domain and the choice of the appropriate mathematical framework for a model depends on the characteristics of the system one wants to study. On the contrary, the structures of models reflect to some extent the structure of the system under study. Therefore, our framework focuses on structural comparisons and is based on S2ML (System Structure Modeling Language) [2].

Models from different engineering domains cannot be compared directly. First, they are abstracted into a pivot language (S2ML). Second, their abstractions are compared. To support model synchronization we develop the SmartSync platform, which is used to compare S2ML abstractions of heterogeneous models.

To illustrate our proposal we use a case study – an Electro-Mechanical Actuator (EMA) of an aileron for a small aircraft. We show how the introduced framework can be used to handle consistency between system architecture models (designed in SysML [5]) and safety models (designed in AltaRica 3.0 [3]) with several alternative system architectures.

This work continues the work on model synchronization presented in [13] and [9]. An interesting study [11] uses model synchronization techniques with hierarchical graphs.

The remainder of this article is organized as follows. Section II introduces the EMA case study. Section III describes the model synchronization framework used to ensure consistency between heterogeneous models. Section IV presents the results. Finally, section V concludes this article and discusses future works.

II. Case study

The considered case-study is an Electro-Mechanical Actuator (EMA) for a general aviation small aircraft. The EMA is intended to actuate the aileron, replacing the usual rod, cables and lever mechanisms. The proposed actuator is driven by the aircraft electrical networks, controlled by the on-board FCC (Flight Control Computers) with a set point consistent with the pilot instructions, taking into account the aileron feedback position and the EMA feedback (Fig.1).

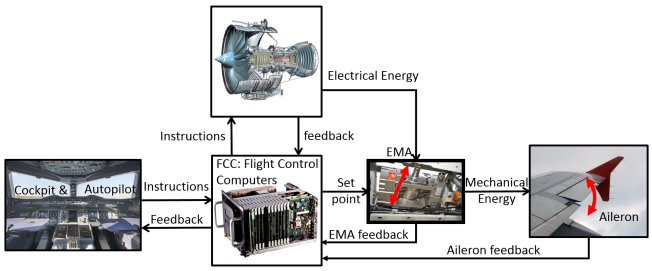

Fig. 1. EMA on-board context.

There are different relevant kinematic architectures such as a 4-bars with a crank and rod mechanism, a 3-bars with an electric cylinder or a direct drive with a motor and a gearbox mounted on the axis of the revolute joint between the wing and the aileron. In this work, we will focus on the 3-bars architecture.

This architecture is illustrated in Fig.2. Linked to the wing and the aileron with two spherical joints, the EMA is made up of a housing that encapsulates all the components, a DC motor controlled by a Micro Controller Unit (MCU) (not represented), a gearbox and a screw and nut assembly to transform the gearbox output rotation into a translation of a rod that will in turn push or pull the aileron.

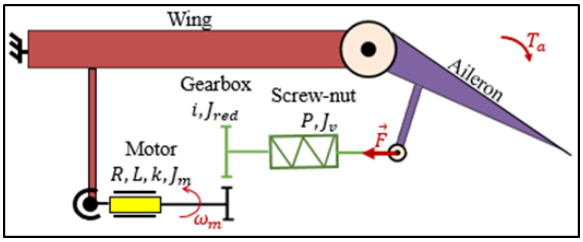

Fig. 2. EMA 3-bars architecture.

III. Model synchronization

A. Principle

Integration of engineering models can be achieved by model synchronization process, i.e. the process by which one can ensure that two possibly heterogeneous models are "speaking" about the same system. Two models, written into two different languages, can generally not be directly compared. The idea is thus to abstract them into a pivot language and to compare their abstractions (see Fig. 3). The synchronization of models goes in three steps. The first step is the abstraction, i.e. the extraction of the common part that can be compared from the models. The second step is the comparison of the abstractions. The third step, the so-called concretization, consists in possible adjustment of the initial models (in case of detected inconsistencies).

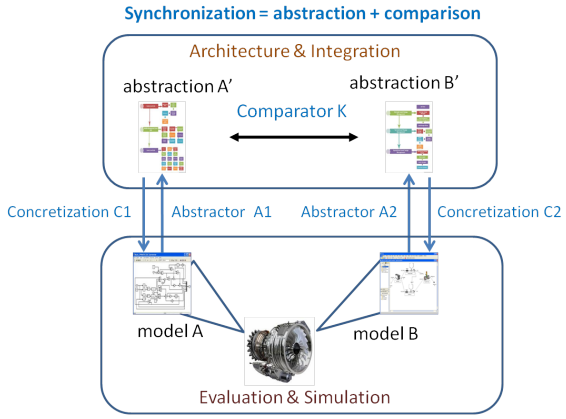

Fig. 3. Model synchronization.

B. S2ML as a pivot language

S2ML (System Structure Modeling Language) [2] aims at providing a structuring paradigm of systems engineering modeling languages. It unifies concepts coming from object-oriented [1] and prototype-oriented [12] programming languages. As heterogeneous models can be essentially compared by their structure, S2ML is a perfect candidate as a pivot language for the abstraction.

1) Basic concepts: S2ML is made of the following basic elements: ports, connections, blocks and attributes. Ports are basic objects of models (e.g. variables, events, parameters). Connections are used to describe relations between ports (e.g. equations, transitions, assertions). Blocks are containers composed of ports, connections and other blocks. Attributes are couples (name = value) used to associate information to ports, connections and blocks.

Example: Consider a non repairable component (*NRComponent*) in AltaRica 3.0 having a Boolean state variable *vsWorking* and a failure event *evFailure*. Its S2ML abstraction would be as illustrated in Fig. 4.

```
class NRComponent
  port vsWorking(kind="variable", type="Boolean",
       init="true");
  port pLambda(kind="parameter", type="Real",
       value="1.0e-5");
  port evFailure(kind="event",
       delay="exponential(pLambda)");
  connection [ evFailure, vsWorking ](type="transition",
       guard="vsWorking", action="vsWorking := false");
end
```

Fig. 4. S2ML code for the block *NRComponent*.

The class *NRComponent* contains three ports *vsWorking*, *evFailure* and *pLambda* having different attributes, and a

connection, which represents the transition labeled by the event *evFailure*. In S2ML, ports, connections and blocks are interpreted by themselves. But a particular modeling language, implementing S2ML as its structuring paradigm, can give a concrete interpretation to ports, connections and blocks. For example, in AltaRica 3.0 variables, parameters, events and observers are interpreted by ports; transitions and assertions are interpreted by connections, blocks and classes are interpreted by blocks and classes.

2) Relations: S2ML introduces several structural relations to build and organize models.

```
class Motor
  extends NRComponent;
  port vfFromMCU (type ="Boolean", reset = "false");
  port vfToGearbox (type="Boolean", reset = "false");
  connection assertion [vfToGearbox, vsWorking, vfFromMCU];
end
class MCU
  extends NRComponent;
  port vfFromElectricPower (type="Boolean", reset="false" );
  port vfFromInstructions (type="Boolean", reset="false" );
  port vfFromIncidenceSensor (type="Boolean",
       reset="false");
  port vfToMotor (type = "Boolean", reset = "false" );
  connection assertion [vfToMotor,
  vsWorking, vfFromElectricPower, vfFromInstructions,
       vfFromIncidenceSensor];
end
block EMASystem_2
  // ports
  block ElectricPower
    extends NRComponent;
    port vfToMCU ( type = "Boolean", reset = "false");
    connection assertion[ vfToMCU, vsWorking];
  end
  block Line1
    embeds main.ElectricPower as EP;
    MCU MCU1;
    Motor Motor1;
    connection assertion[EP.vfToMCU,
         MCU1.vfFromElectricPower];
  end
  clones Line1 as Line2;
  // the remainder of the block EMASystem_2
end
```

Fig. 5. S2ML abstraction of the AltaRica 3.0 model of the EMA system with duplicated MCUs and Motors.

a) Composition: The simplest structural relation is the composition: a system composes a component means that the component "is part of" the system. In S2ML, the composition is represented by adding different components within the code of the system as shown in the example below.

Example: In the example given in Fig. 5, the block *EMASystem_2* contains the blocks *ElectricPower*, *Line1* and *Line2* and also different ports and connections not represented here.

b) Inheritance: Inheritance makes it possible to an element (block or class) to acquire all the properties of another element without explicitly duplicating them. Inheritance implements the "is a" relation between modeling elements. In S2ML, the inheritance is represented by the keyword "extends".

Example: In the AltaRica 3.0 model of the EMA system, all the components extend the class *NRComponent* (see Fig. 4) as they may fail in operation. In the example given in Fig. 5, the block *ElectricPower* extends the class *NRComponent* defined previously. It contains all the ports and connections of the class *NRComponent* and adds a port *vfToMCU* and a connection *assertion*.

c) Prototype/Cloning: A block is a container for ports, connections and other blocks. Each block is a prototype, i.e. it has a unique occurrence in the model. A system may contain similar components or subsystems. To avoid duplicating the description of a block, it is possible to clone an already existing one. In S2ML, the cloning of a block is obtained by the keyword "clones".

Example: In the example given in Fig. 5, the block *EMASystem_2* contains two identical blocks *Line1* and *Line2*, composed of a Motor and a MCU each. In order not to duplicate the code, the block *Line1* is cloned to obtain the block *Line2*.

d) Class/Instance: A second way to avoid duplicating the description of a block consists in declaring a model of the duplicated block in a separate modeling entity, the so-called class, and then in instantiating this class wherever we need to use it again. Obviously, the class is referred to by the keyword "class" in S2ML. Each instance of the class is obtained by writing the name of the class followed by names of the created instances.

Example: In the example given in Fig. 5 two classes *Motor* and *MCU* are defined. They are instantiated inside the block *Line1* of the block *EMASystem_2*.

e) Aggregation: Aggregation is a "uses" relation between modeling components. It makes it possible to represent components which are not a part of the subsystem and may be shared by several subsystems. The clause "embeds" in S2ML refers to an aggregation.

Example: In the example given in Fig. 5, the component `ElectricPower` is used by the *Line1* and the *Line2*. It is aggregated by both subsystems via the clause "embeds".

3) Unfolded model: Any hierarchical model is semantically equivalent to an unfolded (also called instantiated) one. An unfolded S2ML model is a model made of a hierarchy of nested or aggregated blocks, connections and ports. This model is obtained by applying recursively rewriting rules, the so-called unfolding rules. These rules resolve inheritance, classes instantiation, blocks cloning and paths of aggregated elements.

An unfolded (or instantiated) model is used in the comparison step of model synchronization.

Example: Consider the S2ML model given in Fig. 5. The equivalent unfolded block for *EMASystem_2* is presented in Fig. 6.

C. SmartSync platform

The proposed platform for model synchronization SmartSync is illustrated in Fig. 7. The first step of the model synchronization is the abstraction, which consists in translating models into S2ML. This step is still done manually for the moment but it can be automated. In the next step, the abstractions of the different models are compared two by two and a report of these comparisons is generated. This report is then analyzed by the members of the different teams that built the initial models. Together, they produce a matching

```
block EMASystem_2
  // ports
  block ElectricPower
    port vsWorking ( kind = "variable", type="Boolean", init
        = "true" );
    port pLambda (kind ="parameter", type = "Real", value =
        "1.0e-5");
    port evFailure (kind = "event", delay =
        "exponential(pLambda)");
    port vfToMCU ( type = "Boolean", reset = "false");
    connection transition[evFailure, vsWorking];
    connection assertion[vfToMCU, vsWorking];
  end
  block Line1
    embeds main.ElectricPower as EP;
    block MCU1
      port vsWorking(kind="variable", type="Boolean",
          init="true");
      port pLambda(kind="parameter", type="Real",
          value="1.0e-5");
      port evFailure(kind="event",
          delay="exponential(pLambda)");
      port vfFromElectricPower (type="Boolean",
          reset="false" );
      port vfFromInstructions (type="Boolean", reset="false"
          );
      port vfFromIncidenceSensor (type="Boolean",
          reset="false");
      port vfToMotor (type = "Boolean", reset = "false" );
      connection assertion[vfToMotor,
      vsWorking, vfFromElectricPower, vfFromInstructions,
      vfFromIncidenceSensor];
      connection transition[evFailure, vsWorking];
    end
    block Motor1
      port vsWorking (kind="variable", type="Boolean",
          init="true");
      port pLambda (kind="parameter", type="Real",
          value="1.0e-5");
      port evFailure (kind="event",
          delay="exponential(pLambda)");
      port vfFromMCU (type="Boolean", reset="false");
      port vfToGearbox (type="Boolean", reset="false");
      connection transition[evFailure, vsWorking];
      connection assertion [vfToGearbox, vsWorking,
          vfFromMCU];
    end
    connection assertion[main.ElectricPower.vfToMCU,
        MCU1.vfFromElectricPower];
  end
  block Line2
  // the body of the block Line2 (copy of the block Line1)
  end
  // the remainder of the block EMASystem_2
end
```

Fig. 6. Unfolded S2ML model of the EMASystem_2.

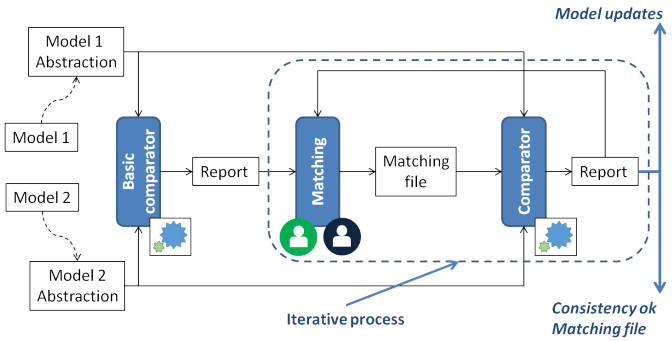

Fig. 7. Models synchronization process.

file that matches the same elements in the two models. When there is no correspondence the keyword ”forget” is used. The next step of the comparison process consists in comparing the initial models using the matching file. Another report is then generated that contains a list of inconsistencies. This report is analyzed again by the members of both teams. The matching file is updated with new corresponding elements. The updated matching file is used again in the comparison of the model abstractions and so on. The process iterates until all the inconsistencies have been resolved. At each iteration, if an inconsistency is detected, one or both models should be updated.

The outcome of the model synchronization is twofold. First, it allows to detect model inconsistencies in which case models need to be updated. Second, it allows to validate the model consistency. Models can then be used to produce performance indicators and so on.

1) Comparison: Different types of comparators (see [14] for an interesting survey on model comparison techniques) for S2ML models can be defined, for instance:

- Dictionary, which consists in matching the names of different elements (ports, nested/aggregated blocks and connections);
- Structural, which consists in matching the names of different elements and the structure of the model;
- Topological, which consists in matching the names of different elements, the structure of the model and the connections between ports.

Note that the choice of abstractors and comparators depends on the system under development and the level of maturity of the project.

The comparison of S2ML models is done as follows. First, S2ML models are unfolded/instantiated, i.e. transformed into a hierarchy of nested/aggregated blocks, ports and connections as described in Section III-B3. Second, the unfolded S2ML models are compared using the matching file. Matching blocks and ports is quite similar. For each block/port of the first model (and vice-versa):

- if there is a correspondence in the matching file (which is different from “forget”) then
  - if the corresponding block/port exists in the second model (and vice-versa) then the consistency is checked;
  - otherwise an inconsistency is detected;
- if the correspondence is “forget” then there is nothing to do;
- otherwise, an inconsistency is detected.

IV. EMA SYSTEM: MODEL SYNCHRONIZATION

In this section we apply model synchronization to the case study presented in Section II. We present a collaborative design of the EMA system. The collaboration is between two teams: system architecture and safety analysis. Each team performs different activities. The first activity is modeling which is performed separately by members of each team using different modeling languages and tools. The second activity is model synchronization, i.e. the verification of consistency between models that ensures that both models are describing the same system. This activity is performed by the members of both teams and involves the SmartSync platform.

Finally, the validated safety model is analyzed, and as the initial architecture is not robust enough, the safety team proposes two alternative architectures to the design team.

A. EMA system variant 1

1) Modeling:

a) System architecture: System architecture models are created using SysML [5] with a particular focus on system physical architecture part.

The internal block diagram representing the first variant (without redundancies) of the EMA physical architecture is given Fig. 8. It has been done using SysML plugin of MagicDraw modeling tool.

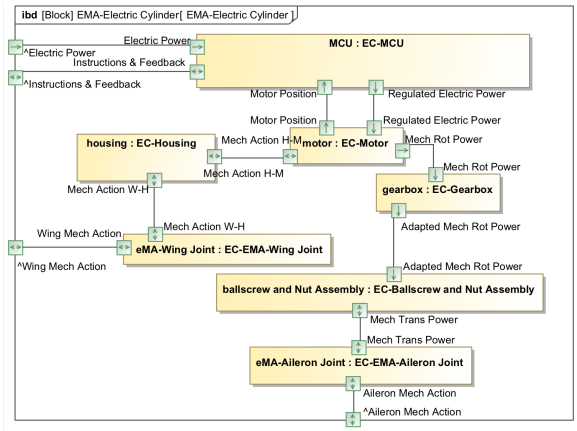

Fig. 8. EMA system physical architecture (variant 1).

This model does not represent the system environment. The incidence sensor is supposed to be a part of the block *Motor*; it is represented by a port *Motor Position* of the block *Motor* connected to the port *Motor Position* of the block *MCU*.

b) Safety: The failure condition of interest is the loss of the aileron incidence control. It can be caused by failures in the EMA itself or failures in the linking joints to the aileron. As this architecture has no redundancies, a single failure of a component leads to the occurrence of the failure condition.

The safety model is created using AltaRica 3.0 modeling language [3] and the OpenAltaRica platform [1]. AltaRica 3.0 is a high level formal modeling language dedicated to safety analyses. It is a textual language but graphical representations can be associated to textual models.

Fig. 9 shows the graphical representation of the AltaRica 3.0 model of the first variant of the EMA system. This model is an extended reliability block diagram, where blocks represent system components and their failures and connections between blocks represent the propagation of failures. The block *Observer* models the failure condition.

2) Synchronization:

[1]https://www.openaltarica.fr/

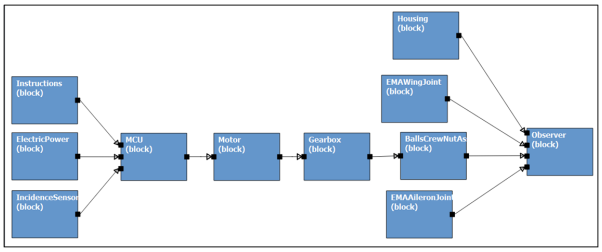

Fig. 9. Graphical representation of the AltaRica 3.0 model of the EMA system (variant 1).

a) Abstraction: First, both models are abstracted, i.e. transformed into S2ML.

For AltaRica 3.0 the transformation is straightforward, as the language uses S2ML as its structural paradigm. State and flow variables, events and parameters are abstracted to S2ML ports; transitions and assertions are transformed into connections; different structural constructs like inheritance, cloning, instantiation, etc. are transformed into their equivalents in S2ML. As an example, an S2ML abstraction of a non repairable component is given Fig. 4.

For SysML internal block diagrams the transformation is also quite simple: parts are transformed into S2ML blocks, ports into S2ML ports and connections between ports are transformed into S2ML connections between corresponding S2ML ports.

b) Comparison: In the next step, the abstractions are compared and a report is generated. This report is analyzed by members of both teams. The following differences are detected:

- Different names of blocks (e.g. the block *BallscrewAndNutAssembly* in the SysML model corresponds to the block *BallsCrewNutAssembly* in the AltaRica 3.0 model);
- Different names of ports (e.g. the port *Motor.RegulatedElectricPower* in the SysML model corresponds to the port *Motor.vfFromMCU* in the AltaRica 3.0 model);
- Elements of system architecture model not represented in the safety model (e.g. *Motor.MechanicalActionHM* has no correspondence in the safety model);
- Elements of the safety model not represented in the system architecture model (e.g. state variables, failure events, parameters, etc. have no equivalent in the system architecture model).

All the differences are listed in the matching file, which makes it possible to establish the correspondence between the two models. Table I shows an extract of a matching file. The first column is the element type (port, block, aggregated block or connection). The second column is the name of the element of the first model, the third column is the name of the corresponding element in the second model. When there is no correspondence, the keyword *forget* is used. It is possible to add a fourth column with comments to justify matching decisions. As we can see in Table I, the block *Observer* of the safety model has no correspondence in the system architecture

TABLE I
EMA SYSTEM ARCHITECTURE AND SAFETY MODELS MATCHING (VARIANT 1), ITERATION 1.

Type	Model1 (SysML)	Model2 (AltaRica 3.0)
block	EMASystem_1	EMASystem_1
port	AileronMechanicalAction	forget
port	ElectricalPower	ElectricPower.vfToMCU
port	InstructionAndFeedback	Instructions.vfToMCU
port	WingMechanicalAction	forget
block	forget	Observer
block	BallScrewAndNutAssembly	BallScrewNutAssembly
block	EMAAileronJoint	EMAAileronJoint
port	AileronMechanicalAction	vfOut
port	MechanicalTransmissionPower	forget
port	forget	evFailure
port	forget	pLambda
port	forget	vsWorking
block	EMAWingJoint	EMAWingJoint
port	MechanicalActionHW	forget
port	WingMechanicalAction	vfOut
port	forget	evFailure
port	forget	pLambda
port	forget	vsWorking
block	Gearbox	Gearbox
...	...	...

model because it represents safety related information (i.e. the failure condition to study). The port *ElectricalPower* in the SysML model corresponds to the port *vfToMCU* of the block *ElectricPower* in the AltaRica 3.0. It is important to note that the block *ElectricPower* of the safety model has no equivalent in the architecture model. In the system architecture model this block is not represented as it belongs to the system environment, whilst the safety analyst decided to represent it in his model because the failure of the electric power causes the occurrence of the failure condition.

The produced matching file is used to compare the abstractions of system architecture and safety models. In the next step of the comparison, new differences are detected. They are analyzed again and the matching file is populated with new matching information summarized in Table II. Models are then compared again. Finally, no more differences are detected. The consistency between system architecture and safety models is verified. The matching file establishes the correspondence between the two models and elements which do not have any correspondence but it is validated by the teams.

TABLE II
EMA SYSTEM ARCHITECTURE AND SAFETY MODELS MATCHING (VARIANT 1), ITERATION 2.

Type	Model1 (SysML)	Model2 (AltaRica 3.0)
block	main.EMASystem_1	main.EMASystem_1
block	BallScrewAndNutAssembly	BallScrewNutAssembly
port	AdaptedMechanicalRotPower	vfFromGearbox
port	MechanicalTransmissionPower	vfToEMAAileronJoint
port	forget	evFailure
port	forget	pLambda
port	forget	vsWorking

3) Analysis: The validated safety model is analyzed by generation of a Fault Tree and calculation of minimal cut sets, which are listed below. As expected, single failures of components lead to the occurrence of the failure condition. The safety team proposes two alternative architectures with redundancies:

V2: two redundant MCUs and two redundant motors;
V3: two redundant MCUs and a double winding motor.

Their models and analyses are presented below.

	Minimal Cut Sets (MCS)
1	ElectricPower.evFailure
2	IncidenceSensor.evFailure
3	Instructions.evFailure
4	MCU.evFailure
5	Motor.evFailure
6	Gearbox.evFailure
7	BallScrewNutAssembly.evFailure
8	EMAAileronJoint.evFailure
9	EMAWingJoint.evFailure
10	Housing.evFailure

B. EMA system variant 2

1) Modeling:

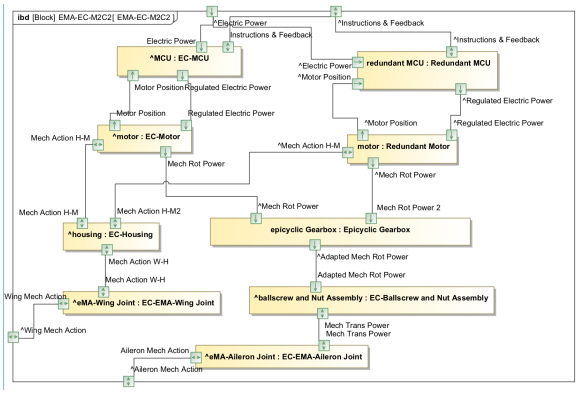

Fig. 10. EMA system physical architecture (variant 2).

a) System architecture: Fig. 10 shows the internal block diagram representing the physical architecture of the second variant of the EMA system. Compared to the first variant of the architecture, it contains two redundant motors (*Motor* and *RedundantMotor*), two redundant MCUs (*MCU* and *RedundantMCU*) and an epicyclic gearbox with two inputs (*MechRotPower* and *MechRotPower2*).

b) Safety: The graphical representation of the corresponding safety model is given in Fig. 11. Compared to the previous AltaRica 3.0 model, it contains:

- two blocks for redundant MCUs (*MCU1* and *MCU2*),
- two blocks for redundant Motors (*Motor1* and *Motor2*),
- two blocks to represent incidence sensors (*IncidenceSensor1* and *IncidenceSensor2*) and
- a new block *Gearbox* with two inputs (*vfFromMotor1* and *vfFromMotor2*).

2) Synchronization: First, both SysML and AltaRica 3.0 models are transformed into S2ML as described previously. Then, the matching file produced for the first variant of the architecture is used directly to compare S2ML models of the

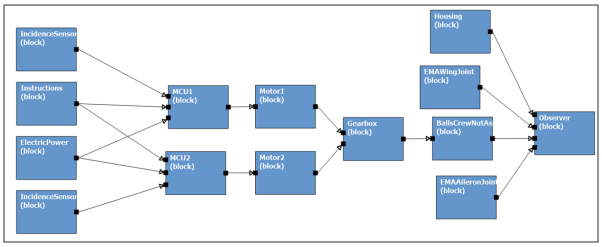

Fig. 11. Graphical representation of the AltaRica 3.0 model of the EMA system (variant 2).

second variant. The report produced by the tool contains only the differences between the two variants.

TABLE III
EMA SYSTEM ARCHITECTURE AND SAFETY MODELS MATCHING (VARIANT 2).

Type	Model1 (SysML)	Model2 (AltaRica 3.0)
block	EMASystem_2	EMASystem_2
block	Gearbox	Gearbox
port	MechanicalRotPower	vfFromMotor1
port	MechanicalRotPower2	vfFromMotor2
block	MCU1	MCU1
port	ElectricalPower	vfFromElectricPower
port	InstructionAndFeedback	vfFromInstructions
port	MotorPosition	vfFromIncidenceSensor
port	RegulatedElectricalPower	vfToMotor
port	forget	evFailure
port	forget	pLambda
port	forget	vsWorking
block	MCU2	MCU2
...	...	...
block	Motor1	Motor1
port	MechanicalAction	forget
port	MechanicalRotPower	vfToGearbox
port	MotorPosition	IncidenceSensor1.vfToMCU
port	RegulatedElectricalPower	vfFromMCU
port	forget	evFailure
port	forget	pLambda
port	forget	vsWorking
block	Motor2	Motor2
...	...	...

The matching file is populated with the new matching information from the Table III. It is used to compare again the abstractions. No more differences are detected. The consistency between system and safety models of the second variant of the EMA system is verified. As we can see, the reuse of matching file of the first variant of the architecture greatly simplifies the consistency verification for the second variant of the architecture.

3) Analysis: The validated safety model is analyzed by generation of a Fault Tree and calculation of minimal cut sets, which are listed below. This architecture is more robust. Single point failures remain only on the mechanical and joint-attach part of the system. But another variant of the architecture also needs to be evaluated.

	Minimal Cut Sets (MCS)
1	ElectricPower.evFailure
2	Instructions.evFailure
3	Gearbox.evFailure
4	BallScrewNutAssembly.evFailure
5	EMAAileronJoint.evFailure
6	EMAWingJoint.evFailure
7	Housing.evFailure
8	IncidenceSensor1.evFailure, IncidenceSensor2.evFailure
9	IncidenceSensor1.evFailure, MCU2.evFailure
10	IncidenceSensor1.evFailure, Motor2.evFailure
11	IncidenceSensor2.evFailure, MCU1.evFailure
12	MCU1.evFailure, MCU2.evFailure
13	MCU1.evFailure, Motor2.evFailure
14	IncidenceSensor2.evFailure, Motor1.evFailure
15	MCU2.evFailure, Motor1.evFailure
16	Motor1.evFailure, Motor2.evFailure

C. EMA system variant 3

1) Modeling:

a) System architecture: Fig. 12 shows the internal block diagram representing the physical architecture of the third variant of the EMA system. Compared to the first variant of the architecture, it contains two redundant MCUs (*MCU* and *RedundantMCU*) and a double winding motor (*DoubleWindingMotor*) composed of two motor windings (*MotorWinding1* and *MotorWinding2*) and a motor shaft (*MotorShaft*).

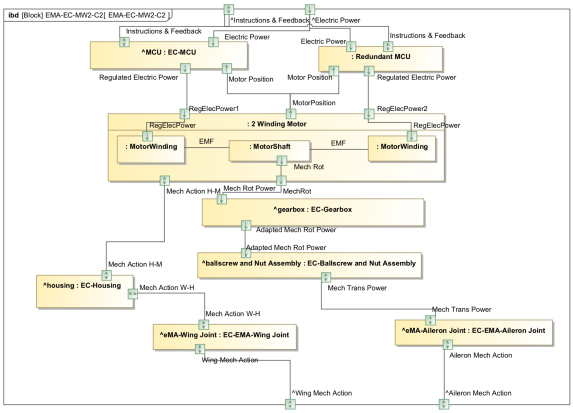

Fig. 12. EMA system physical architecture (variant 3).

b) Safety: Graphical representation of the corresponding safety model is given in Fig. 13. Compared to the AltaRica 3.0 model of the first variant, it contains:

- two blocks for redundant MCUs (*MCU1* and *MCU2*), and
- a new block *Motor* with two inputs (*vfFromMCU1* and *vfFromMCU2*) composed of two blocks representing windings and their failures (*Winding1* and *Winding2*).

2) Synchronization: First, both SysML and AltaRica 3.0 models are transformed into S2ML as explained previously. Then, the matching file produced for the first variant of the architecture is used directly to compare S2ML models of the third variant. System architect and safety analyst only need to match the differences between the two variants.

After the first iteration, the results of matching activity are given in Table IV. The block *DoubleWindingMotor.MotorShaft*

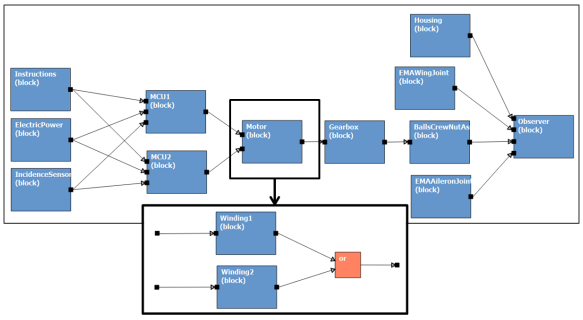

Fig. 13. Graphical representation of the AltaRica 3.0 model of the EMA system (variant 3).

TABLE IV
EMA SYSTEM ARCHITECTURE AND SAFETY MODELS MATCHING (VARIANT 3).

Type	Model1 (SysML)	Model2 (AltaRica 3.0)
block	EMASystem_3	EMASystem_3
block	DoubleWindingMotor	Motor
port	MechanicalAction	forget
port	MechanicalRotPower	vfToGearbox
port	MotorPosition	IncidenceSensor.vfToMCU
port	RegulatedElectricalPower1	vfFromMCU1
port	RegulatedElectricalPower2	vfFromMCU2
block	MotorShaft	
block	MotorWinding1	Winding1
block	MotorWinding2	Winding2

does not have any correspondence in the safety model. In this case, an inconsistency is detected in the safety model: failure of the motor shaft should be taken into account. The safety model needs to be updated.

D. Summary

The conducted experience shows that models produced by different systems engineering disciplines are different but it is possible to establish a correspondence between them. The correspondence is saved in the so-called matching file, which is populated iteratively. It can also be reused for model synchronization of new variants of the system architecture which greatly simplifies the consistency verification. As we can see, model synchronization makes it possible not only to check consistency between models but also to detect inconsistencies.

V. Conclusion and Perspectives

In this article, we presented experiments on the synchronization of system architecture and safety models of an electromechanical actuator for an aileron of a small aircraft. We showed that model synchronization can be used to ensure the consistency of heterogeneous models, designed within different formalisms and different modeling environments.

To support model synchronization, we developed the SmartSync platform, which relies on S2ML as pivot language. With SmartSync, we studied several alternative architectures of the EMA system. We checked consistency between architecture and safety models for the first and second variants and detect an inconsistency in the safety model of the third variant of architecture.

The process of making models consistent is iterative and involves representatives of the engineering disciplines at stake. The SmartSync platform helps not only to check the consistency between models, but also to detect inconsistencies within models and to support the dialog between stakeholders.

As future works, we plan to improve SmartSync, notably by developing new comparison algorithms and abstraction methods.

References

[1] Mauricio Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, New-York, USA, 1998.

[2] M. Batteux, T. Prosvirnova, and A.Rauzy. From models of structures to structures of models. In *4th IEEE International Symposium on Systems Engineering, ISSE 2018*, Rome, Italy, October 2018.

[3] M. Batteux, T. Prosvirnova, and A.Rauzy. Altarica 3.0 in 10 modeling patterns. *International Journal of Critical Computer-Based Systems (IJCCBS)*, 9:133, 2019.

[4] Pierre David, Vincent Idasiak, and Frederic Kratz. Reliability study of complex physical systems using sysml. *Reliability Engineering & System Safety*, 95(4):431–450, 2010.

[5] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann. The MK/OMG Press, San Francisco, CA 94104, USA, 2011.

[6] Peter Fritzson. *Principles of ObjectOriented Modeling and Simulation with Modelica 3.3: A CyberPhysical Approach*. Wiley-IEEE Press, Hoboken, NJ 07030-5774, USA, 2015.

[7] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[8] Harold Klee and Randal Allen. *Simulation of Dynamic Systems with MATLAB and Simulink*. CRC Press, Boca Raton, FL 33431, USA, February 2011.

[9] Anthony Legendre, Agnes Lanusse, and Antoine Rauzy. Toward Model Synchronization Between Safety Analysis and System Architecture Design in Industrial Contexts. In Yiannis Papadopoulos Marco Bozzano, editor, *Model-Based Safety and Assessment*, volume 10437, pages 35–49. Springer, 2017. Proceedings of the 5th International Symposium, IMBSA 2017, Trento, Italy, September 11–13, 2017,.

[10] Pierre Mauborgne, Samuel Deniaud, Eric Levrat, Eric Bonjour, Jean-Pierre Micaëlli, and Dominique Loise. Operational and system hazard analysis in a safe systems requirement engineering process application to automotive industry. *Safety Science*, 87:256–268, August 2016.

[11] S. Missaoui, F. Mhenni, J. Choley, and N. Nguyen. Verification and validation of the consistency between multi-domain system models. In *2018 Annual IEEE International Systems Conference (SysCon)*, pages 1–7, April 2018.

[12] James Noble, Antero Taivalsaari, and Ivan Moore. *Prototype-Based Programming: Concepts, Languages and Applications*. Springer-Verlag, Berlin and Heidelberg, Germany, 1999.

[13] Tatiana Prosvirnova, Estelle Saez, Christel Seguin, and Pierre Virelizier. Handling consistency between safety and system models. In *IMBSA 2017 (International Symposium on Model-Based and Assessment)*, pages pp. 19–34, Trento, Italy, September 2017.

[14] Matthew Stephan and James R. Cordy. A survey of model comparison approaches and applications. In *MODELSWARD 2013 - Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development, Barcelona, Spain, 19 - 21 February, 2013*, pages 265–277, 2013.

[15] Nataliya Yakymets, Yupanqui Munoz Julho, and Agnes Lanusse. Sophia framework for model-based safety analysis. In *Actes du congrès Lambda-Mu 19 (actes électroniques)*, Dijon, France, October 2014. Institut pour la Maîtrise des Risques.