Model synchronization: a formal framework for the management of heterogeneous models

Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy

Model synchronization: a formal framework for the management of heterogeneous models

Michel Batteux[1], Tatiana Prosvirnova[2,3], and Antoine Rauzy[4]

[1] IRT SystemX, Palaiseau, France
michel.batteux@irt-systemx.fr
[2] Laboratoire Genie Industriel, CentraleSupélec, Gif-sur-Yvette, France
[3] ONERA/DTIS, UFTMiP, Toulouse, France,
tatiana.prosvirnova@onera.fr
[4] Norwegian University of Science and Technology, Trondheim, Norway
antoine.rauzy@ntnu.no

Abstract. In this article, we present the conceptual foundations and implementation principles of model synchronization, a formal framework for the management of heterogeneous models. The proposed approach relies on S2ML (System Structure Modeling Language) as a pivot language. We show, by means of a case study, that model synchronization can be used to ensure the consistency between system architecture models designed with Capella and safety models written in AltaRica 3.0.

Keywords: Heterogeneous models · model synchronization · S2ML.

1 Introduction

To face the increasing complexity of technical systems, systems engineers are designing models. These models serve different purposes: system architecture, control engineering, multi-physics simulation, safety analyses, performance assessments. They are designed at different levels of abstraction and by different teams. They may have also different levels of maturity. Ensuring that these models are consistent one another is one of today's major industrial challenges. As of today, their integration relies almost exclusively on organizational processes.

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. Different model transformation techniques have been proposed (e.g. [12], [18]) but they often assume a master/slaves organization of models, which is not realistic in practice. As an interesting alternative, two-sided model transformation based on triple graph grammars has been proposed see e.g. [9].

In this article, we present the conceptual foundations and implementation principles of model synchronization, a formal framework to ensure the consistency of heterogeneous models. Model synchronization relies on ideas stemmed from Cousot's abstract interpretation [6], but its implementation is dedicated to the problem at stake. Namely, the overall approach relies on four theses:

Thesis 1. The diversity of models is irreducible. Moreover, each model has its own life-cycle. In other words, attempts to derive models for one purpose (e.g. safety analyses) from models designed for another purpose (e.g. system architecture), are essentially vain and even counter-productive.

Thesis 2. Heterogeneous models cannot be compared directly. Therefore, the synchronization process is made of three steps: first, models are abstracted in a common language; second, their abstractions are compared; third, actions are possibly taken to adjust original models according to findings of the comparison.

Thesis 3. Systems engineering models are made of two types of constructs: behavioral descriptions and structuring constructs. Behavioral descriptions are specific to each engineering domain. It is thus in general impossible to perform cross-domain comparisons. On the contrary, the structures of models reflect to some extent the structure of the system under study. Therefore, model synchronization focuses on structural comparisons.

Thesis 4. The overall objective of model synchronization is not to reach a perfect matching between (the structures of) original models. Rather, it is to agree on disagreements and to trace the possible discrepancies.

In a word, model synchronization is a pragmatic approach providing a formal framework and concrete tools to improve current processes. Its implementation relies on three basic constituents: first, one needs a pivot language in which models are abstracted. S2ML (System Structure Modeling Language) [2] is an excellent candidate for this purpose as it gathers in an organized and unified way most of the structuring constructs found in systems engineering modeling formalisms. Second, one needs tools to abstract original models into the pivot language. Ideally, the abstraction process should be fully automated. It is possible however to do this part of the work by hand or in a semi-automated way. Finally, one needs software tools to compare abstractions. The development of these tools is justified for at least two reasons: first, they depend only on the pivot language and are therefore reusable for the synchronization of any type of models; second, they ensure the soundness, the completeness and the traceability of the comparison process.

The contribution of this article is thus to present model synchronization and to discuss its conceptual foundations and its implementation into the SmartSync platform. We illustrate the discussion by applying the proposed approach on a case study – an electrical power supply system borrowed from [5]. We show how it can be used to maintain the consistency between system architecture models designed with Capella [16] and safety models written in AltaRica 3.0 [3].

The remainder of this article is organized as follows. Section 2 introduces the case study. Section 3 describes the model synchronization process. Section 4 discusses model comparison. Section 5 presents the SmartSync platform and gives some experimental results. Finally, section 6 concludes this article and discusses future works.

2 Illustrative Example

2.1 Description

As an illustrative example, we shall consider a power supply system borrowed from [5] and pictured Fig. 1. We shall use this case study to illustrate the different concepts of model synchronization, i.e. to show how to ensure consistency of heterogeneous models.

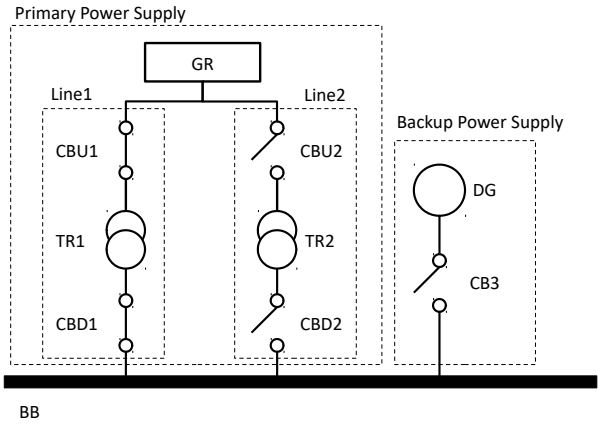

Fig. 1. A power supply system

This system is in charge of supplying electrical power to the busbar BB. It is divided into a primary power supply or a backup power supply. The primary power supply receives the power from the grid and is itself made of two redundant lines. Each of lines is made of a transformer TRi and two circuit breakers CBUi and CBDi, $i = 1, 2$. Lines 1 and 2 are used in alternation. The passive one is in cold redundancy with the active one. The backup power supply part is made of the diesel generator DG and the circuit breaker CB3. It is in cold redundancy with the primary power supply.

2.2 Models

We consider this system from two engineering point of views: the point view of the system architect, supported by models designed in Capella [16], and the point view of the safety analyst, supported by models written in AltaRica 3.0 [3].

Fig. 2 shows the functional architecture diagram of the power supply system, while Fig. 3 presents its physical architecture diagram. The latter is quite similar to the process and instrumentation diagram showed Fig. 1. Fig. 4 on the left represents the life-cycle diagram of the operational architecture.

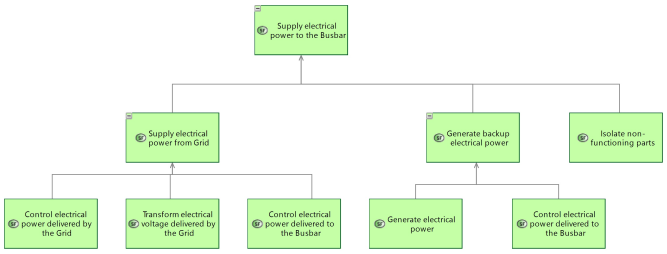

Fig. 2. Functional architecture of the power supply system represented with Capella.

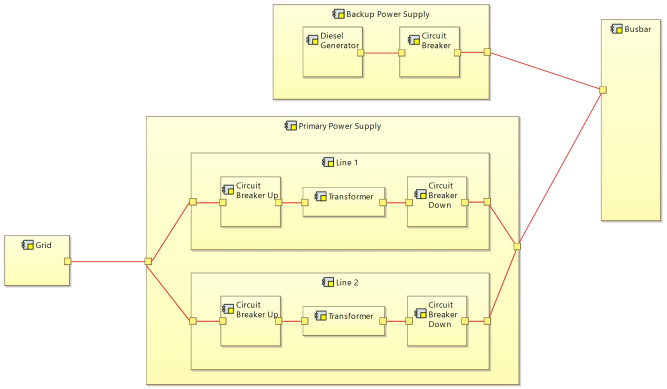

Fig. 3. Physical architecture of the power supply system represented with Capella.

Table 1 summarizes the allocation between functions and physical components depending on different operational phases of the system. Phase 1 corresponds to the mode *Line1* of the diagram Fig. 4, Phase 2 corresponds to the mode *Line2*, and Emergency mode – to the mode *Backup*.

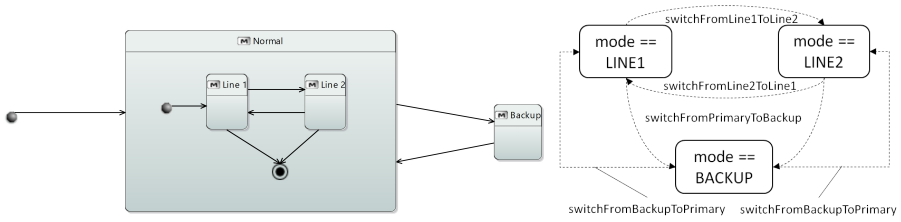

Fig. 4. Capella life-cycle diagram (operational architecture) and graphical representation of the AltaRica 3.0 controller of the power supply system.

Table 1. Power supply system: functions allocation table

Phase	Control electrical power delivered by the Grid	Transform electrical voltage delivered by the Grid	Control electrical power delivered to the Busbar	Generate backup electrical power	Control power delivered to Busbar	Isolate non-functioning parts
Phase 1 (Line1)	CBU1	TR1	CBD1			CBU2, CBD2, CB3
Phase 2 (Line2)	CBU2	TR2	CBD2			CBU1, CBD1, CB3
Emergency mode (Backup)				DG	CB3	CBU1, CBD1, CBU2, CBD2

Fig. 5 shows an excerpt of the AltaRica code for the power supply system. There are two failure conditions of interest: loss of electrical power delivered to the busbar and loss of isolation (of non-functioning parts). They are represented by two observers in the AltaRica model.The structure of the model is inspired by the phased-mission systems modeling pattern [4] and is close to the structure of the Capella model. The block *Controller*, which graphical representation is sketched in Fig. 4 on the right, corresponds to the life-cycle diagram given Fig. 4 on the left, the block *Functional* – to the functional architecture diagram given Fig. 2 and the block *Physical* to the physical architecture diagram given Fig. 3. The allocation of functions (see Table 1) is represented by the aggregation relation ("embeds" clause). For instance, the function *SupplyPowerByGrid* aggregates the grid, the circuit breakers and the transformer of the Line 1 of the primary power supply system in the phase 1.

System architecture and safety analyses can be seen as two faces of the same medal. System architecture focuses on how the system works, what it should do and should be. It is ruled by so-called architectural frameworks such as the CESAM framework [11]. Safety analyses focus on how the system may fail and what are the consequences of failures.

Although they consider the system at about the same level of abstraction, models designed by system architects and safety analysts are quite different. In particular, the former are pragmatic while the latter are formal [15], two characteristics that we shall define formally in the next section. Ensuring the consistency of these models is thus both extremely important and far from easy.

```
domain MODE {LINE1, LINE2, BACKUP}
block PowerSupplySystem
  block Controller
    // body of the block Controller
  end
  block Functional
    block SupplyElectricalPowerToBusbar
      block SupplyPowerByGrid
        block Phase1
          embeds main.Physical.PrimaryPowerSupply.GR as GR;
          embeds main.Physical.PrimaryPowerSupply.Line1.CBIn as CBU1;
          embeds main.Physical.PrimaryPowerSupply.Line1.TR as TR1;
          embeds main.Physical.PrimaryPowerSupply.Line1.CBOut as CBD1;
          Boolean vfFailed (reset = true);
          assertion
            vfFailed := GR.vfFailed or TR1.vfFailed or CBU1.vfFailedToClose or
                  CBD1.vfFailedToClose;
        end
        // the remainder of the block SupplyPowerByGrid
      end
      // the remainder of the block SupplyElectricalPowerToBusbar
    end
    // the remainder of the block Functional
  end
  block Physical
    block PrimaryPowerSupply
      Grid GR;
      block Line1
        embeds owner.GR as GR;
        Boolean vfInflow, vfOutflow, vfFailed (reset = false);
        CircuitBreaker CBIn, CBOut;
        Transformer TR;
        // the remainder of the block Line1
      end
      clones Line1 as Line2;
      Boolean vfOutflow (reset = false);
      assertion
        Line1.vfInflow := GR.vfOutflow;
        Line2.vfInflow := GR.vfOutflow;
        vfOutflow := Line1.vfOutflow or Line2.vfOutflow;
    end
    // the remainder of the block Physical
  end
  observer Boolean LossOfBusbarPowerSupply = if (Controller.mode==LINE1) then
        Functional.SupplyPowerByGrid.Phase1.vfFailed else if (Controller.mode==LINE2) then
        Functional.SupplyPowerByGrid.Phase2.vfFailed
         else Functional.BackupSupply.EmergencyMode.vfFailed;
  observer Boolean LossOfIsolation = if (Controller.mode==LINE1) then
        Functional.IsolateNonFunctioningParts.Phase1.vfFailed else if
        (Controller.mode==LINE2) then Functional.IsolateNonFunctioningParts.Phase2.vfFailed
        else Functional.IsolateNonFunctioningParts.EmergencyMode.vfFailed;
end
```

Fig. 5. Excerpt of the AltaRica code for the power supply system.

3 Model Synchronization

3.1 Models = Behaviors + Structures

Ensuring the consistency of two or more heterogeneous models requires to understand the nature and the role of each of these models. Models involved in systems engineering serve actually very different purposes. They can be roughly separated into two categories [15]:

– Pragmatic models that are used primarily to support the communication amongst stakeholders.

– Formal models that are used primarily to calculate indicators or to perform simulations.

The latter encode eventually mathematical objects. Their syntax and their semantics must be perfectly defined. They are written in modeling languages such as Modelica [8], Matlab Simulink [10] or AltaRica [3]. On the contrary, the former can only be understood by referring to the system under study. They are often written in standardized graphical notations such as SysML [7] or Capella [16]. For this reason, they have no formal syntax and even less a formal semantics.

Note that formal languages could be used to design pragmatic models (the reverse is indeed not true). However, there is an epistemic gap between pragmatic and formal models: as the former aim primarily at supporting the communication, they keep a lot of knowledge implicit. Making this knowledge explicit would overload them uselessly. Even if we restrict our attention to formal models, their underlying mathematical frameworks can be very different, e.g. systems of ordinary differential equations for Modelica and Simulink and guarded transition systems for AltaRica. This is the reason why, comparing behaviors described by heterogeneous models is essentially meaningless: the comparison should focus on the structural part of models.

Systems engineering modeling formalisms and languages are actually made of two parts: an underlying mathematical model, that aims at describing behaviors, and a structuring paradigm that makes it possible to organize models, i.e. to design them by assembling parts into hierarchical descriptions. The structural parts of SysML and Capella are stemmed from prototype-oriented programming [13], although without clearly acknowledging it. Modelica and Simulink rely on object-orientation. AltaRica 3.0 relies on a combination of both.

3.2 Model Synchronization Principle

As already said, two models, possibly written into two different languages, cannot in general be compared directly, see [17] for an interesting survey on model comparison techniques. Therefore, the synchronization process is made of three steps: first, models are abstracted in a common language; second, their abstractions are compared; third, actions are possibly taken to adjust original models according to findings of the comparison. This third step is called concretization, according to the abstract interpretation terminology. This process is illustrated Fig. 6.

It is worth noticing that different abstractors and comparators can be defined. The choice of the abstractors and the comparators to apply depends on the system under study and the level of maturity of the project.

3.3 System structure modeling language (S2ML)

Describing the structure of a system is fully part of the modeling process. It helps to design, to share, to maintain and eventually to synchronize models.

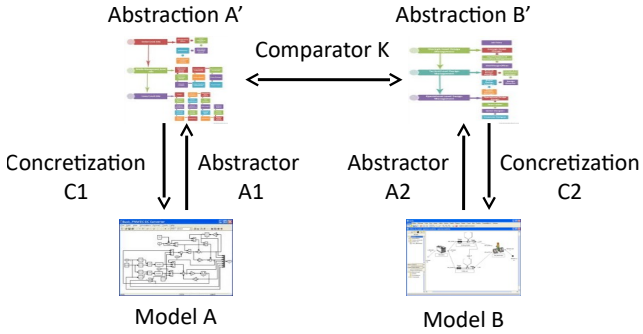

Fig. 6. Model synchronization: principle.

S2ML aims at providing a structuring paradigm of systems engineering modeling languages. It gathers and unifies concepts from object-orientation [1] and prototype-orientation [13]. Due to space limitations, we shall only sketch here S2ML ideas. The reader interested in a more detailed presentation should refer to our article [2].

As heterogeneous models can be essentially compared by their structure, S2ML is a perfect candidate as a pivot language for the abstraction.

S2ML relies on only eight constructs:

- Three types of basic objects: ports, connections and blocks.
- Three structural relations: composition, inheritance and aggregation.
- Two mechanisms making possible to reuse modeling elements: the prototype/clone and the class/instance mechanisms.

Ports are basic objects of models, e.g. variables, events, parameters. They have a basic type such as Boolean, integer, real or some enumerated value.

Connections are used to describe relations between ports, e.g. equations, transitions, assertions.

Blocks are containers to compose ports, connections and other blocks. They are prototypes in the sense of object-oriented theory.

Attributes are pairs (name, value) used to associate information to ports, connections and blocks.

The most important and the simplest structural relation is the composition: a container (prototype or class) composes an element if this element "is part of" the container. The inheritance and the aggregation are respectively "is-a" and "uses" types of relation.

Prototypes and classes are containers. As suggested by their names, prototypes have *a priori* a unique occurrence in the model. It is however possible to clone a prototype. Classes are on-the-shelf, reusable modeling elements. Strictly speaking, they are not part of the models. Rather, they are instantiated into models. Respective advantages and drawbacks of prototypes and classes are discussed in reference [2].

The S2ML+X paradigm consists in designing domain specific modeling languages as the combination of S2ML with a given underlying mathematical framework (the X). We applied already this principle to design AltaRica 3.0, but also to design languages for constraint solving and combinatorial optimization, Boolean reliability models, hierarchical graph representations, hierarchical Markov chains and process algebras (themselves used to describe business processes).

In the S2ML+X paradigm, models are seen as scripts. S2ML provides commands to declare modeling elements. The actual model is obtained by executing these commands. This process works in two steps:

- First, the model is rewritten into a hierarchy of blocks. Each block of the hierarchy may compose ports and connections. This step is called instantiation in the S2ML jargon.
- Second, the hierarchy is removed to get a model made of only one block composing ports and connections. This step is called flattening in the S2ML jargon.

In the framework of model synchronization, the rewriting process is stopped after instantiation, as we are interested in keeping hierarchical, i.e. structural, information.

Note that the abstraction of original models into S2ML models can vary significantly from one model to the other one. It depends on the objectives of the synchronization as well as on the modeling formalism used to design the source model.

4 Model Comparison

A key step of model synchronization consists in comparing the two instantiated S2ML models.

4.1 Instantiated S2ML models

S2ML models to be compared are instantiated, i.e. they are made of three types of objects: ports, connections and blocks. Ports and blocks are uniquely identified by their name. Connections are structured terms involving constants, ports and operators. They may also have some attributes. However, they are just considered as (anonymous) sets of ports in the comparison process. Finally, blocks can compose ports, connections and other blocks. All objects may have some attributes but we shall not consider them here. A model is just a block, possibly rooting a hierarchy of blocks.

Formally, a model is thus a quintuple $\langle P, C, B, \ltimes, r\rangle$ where:

- P and B are two disjoint finite sets of symbols called respectively ports and blocks.
- C is a multiset of connections, i.e. of subsets of P.
- $\ltimes$ is a composition relation, i.e. a subset of $B \times (P \cup C \cup B)$ verifying:

- For each object $o \in P \cup C \cup B$, there exists at most one block $b \in B$ such that $b \ltimes o$. b is called the parent of o.
- $r \in B$ is the unique block with no parent, moreover for all object $o \in P \cup C \cup B$, $r \ltimes^\star o$, where $\ltimes^\star$ denotes the transitive closure of $\ltimes$.

We denote by $\mathcal{M}$ the set of instantiated S2ML models defined as above.

4.2 Matchings

We can now define mappings from models to models. For the sake of model comparisons, we are especially interested in structure preserving mappings.

A mapping α from the model $M : \langle P_M, C_M, B_M, \ltimes_M, r_M \rangle$ to the model $N : \langle P_N, C_N, B_N, \ltimes_N, r_N \rangle$ is structure preserving if the following conditions hold.

- For any block $b \in B_M$ and any object $o \in P_M \cup C_M \cup B_M$, $b \ltimes_M o \Rightarrow \alpha(b) \ltimes_N \alpha(o)$.
- For any connection $c = \{p_1, \ldots, p_k\} \in C_M$, $\alpha(c) \supseteq \{\alpha(p_1), \ldots, \alpha(p_k)\}$, moreover for all $p \in P_M$, if $p \notin c$ then $\alpha(p) \notin \alpha(c)$.

A structure preserving mapping is injective if the following condition holds.

- For any two objects $o, o' \in P_M \cup C_M \cup B_M$, $o \neq o' \Rightarrow \alpha(o) \neq \alpha(o')$.

Injective structure preserving mappings can be seen as projections.

Now, let M, N_1 and N_2 be three models. N_1 and N_2 are matched by M if there exist two injective structure preserving mappings $\alpha_1 : M \to N_1$ and $\alpha_2 : M \to N_2$. The model M catches the commonalities between N_1 and N_2. Building such models M is the objective of the comparison process.

Note that instantiated S2ML models together with structure preserving mappings form a category, see e.g. [14] for an introduction. The notion of matching defined here is inspired from the notion of pullback in category theory.

5 Experiments

5.1 SmartSync platform

The SmartSync platform supports model synchronization. It is based on S2ML as a pivot language for the abstraction. It works as illustrated Fig. 7. The objective is to check the consistency of two models of the same system possibly written in two different languages.

This works in three phases.

The first phase consists in abstracting original models into S2ML. As of today, this is done manually but this could be automated.

The next phase consists in comparing model abstractions. It involves designers of both models. It aims at establishing a structure preserving matching between the elements of the two abstract models This matching is concretely encoded by means of a two columns table (one per model). In a first step, elements

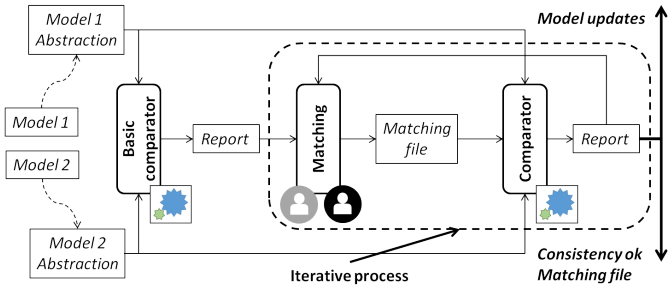

Fig. 7. Models synchronization process.

are automatically matched by traversing down the structure of each model and according to identifiers. Elements that could not be matched are highlighted. It is then possible to match elements "by hand". It is also possible to indicate that an element should not be matched because it is specific to its model. The automatic matching process is then launch again. This process is iterated until no progress can be done anymore.

At the end of the second phase, a (possibly empty) list of inconsistencies is obtained. This list is the input for the third phase, which consists in doing some "homework" on each original model so to solve the problems.

The whole process can be itself iterated.

5.2 Case study: a power supply system

We apply our model synchronization framework to the case study presented in Section 2. We present a collaborative design of the power supply system. The collaboration is between two teams: system architecture and safety analyses. Each team performs different activities.

The first activity is modeling which is performed independently by members of both teams using different modeling languages and tools. System architecture models designed with Capella and safety models written in AltaRica 3.0 are given in Section 2.2.

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.

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.

For Capella functional and physical architecture diagrams the transformation is also quite simple: blocks are transformed into S2ML blocks, ports into S2ML

ports and connections between ports are transformed into S2ML connections between corresponding S2ML ports. The allocation table (see Table 1) is transformed as follows: each functional S2ML block aggregates (via the "embeds" clause) the corresponding allocated physical S2ML blocks.

In the next step, the abstractions are compared and a report is generated. This report is analyzed by members of both teams. All the differences are listed in the matching file, which makes it possible to establish the correspondence between the two models. Table 2 shows the matching file of the first iteration. The

Table 2. Power supply system architecture and safety models matching, iteration 1.

Type	**Model1 (Capella)**	**Model2 (AltaRica 3.0)**
block	SystemArchitecture	PowerSupplySystem
port	forget	LossOfBusbarPowerSupply
port	forget	LossOfIsolation
block	FunctionalPart	Functional
block	OperationalPart.StateMachine	Controller
block	PhysicalPart	Physical

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. The following differences are detected:

- Different names of blocks (e.g. the block *FunctionalPart* in the Capella model corresponds to the block *Functional* in the AltaRica 3.0);
- Elements of the safety model not represented in the system architecture (e.g. observers *LossOfBusbarPowerSupply* and *LossOfIsolation* represent the failure conditions and do not have any equivalent in the Capella model).

The produced matching file is used to compare again the abstractions of the system architecture and safety models. In the next iteration of the comparison, new differences are detected. They are analyzed again and the matching file is populated with new matching information summarized in Table 3. Other differences are detected:

- Different names of ports (e.g. the port *Busbar.input* in the Capella model corresponds to the port *Busbar.vfInflow* in the AltaRica 3.0 model);
- Elements of system architecture model not represented in the safety model (e.g. the port *PhysicalPart.input* has no correspondence in the safety model);
- Different structural decomposition (e.g. in the Capella model the block *Grid* belongs to the block *PhysicalPart* whilst in the AltaRica 3.0 it belongs to the block *PrimaryPowerSupply*).

Table 3. Power supply system architecture and safety models matching, iteration 2.

Type	**Model1 (Capella)**	**Model2 (AltaRica 3.0)**
block	SystemArchitecture	PowerSupplySystem
block	FunctionalPart	Functional
block	SupplyElectricalPowerToBusbar	SupplyElectricalPowerToBusbar
block	GenerateBackupElectricalPower	BackupSupply
block	SupplyElectricalPowerFromGrid	SupplyElectricalPowerByGrid
...	...	...
block	PhysicalPart	Physical
block	Busbar	Busbar
port	input	vfInflow
block	Grid	PrimaryPowerSupply.GR
port	input	forget
port	output	vfOutflow
block	BackupPowerSupply	BackupPowerSupply
port	output	vfOutflow
port	forget	vfFailed
block	DieselGenerator	DG
block	CB	CB
port	input	vfInflow
port	output	vfOutflow
port	forget	fail_close
...	...	...
block	PrimaryPowerSupply	PrimaryPowerSupply
block	Line1	Line1
port	input	vfInflow
port	output	vfOutflow
port	forget	vfFailed
block	CBD	CBOut
block	CBU	CBIn
...	...	...

As we can see it is quite simple to establish the correspondence between system physical architecture *PhysicalPart* and the block *Physical* of the safety model: each Capella block has a corresponding block in the AltaRica 3.0 model, almost each port of the Capella model has a corresponding port in the AltaRica 3.0 model, there are ports in the AltaRica 3.0 model which do not have any correspondence in the Capella model (state variables, events, some flow variables representing failures). Obviously, it is possible in the abstraction step of the safety model not to consider state variables and events as they represent the internal behavior of components and are not expected to have any equivalence in the architecture model.

Concerning the operational part, it is not so obvious to establish the correspondence between the state chart diagram given Fig. 4 on the left and the AltaRica 3.0 model of the *Controller* sketched in the same figure on the right.

For the functional part, the correspondence is not so easy: the functional decomposition of the architecture model is finer than that of the safety model. However, the established correspondence is given in the following table.

Type	**Model1 (Capella)**	**Model2 (AltaRica 3.0)**
block	SystemArchitecture	PowerSupplySystem
block	FunctionalPart	Functional
block	SupplyElectricalPowerToBusbar	SupplyElectricalPowerToBusbar
block	SupplyElectricalPowerFromGrid	SupplyPowerByGrid
block	ControlElectricalPowerDeliveredByGrid. Phase1	Phase1
block	ControlElectricalPowerDeliveredToBusbar. Phase1	Phase1
block	TransformElectricalVoltageDeliveredByGrid. Phase1	Phase1
...	...	...

Models are then compared again. When no more differences are detected, the structural consistency between system architecture and safety models is verified. The matching file establishes the correspondence between the two models. In case of inconsistencies detection, the initial models need to be adjusted.

6 Conclusion

In this article, we presented model synchronization – a formal framework for management of heterogeneous models. This framework is based on S2ML (System Structure Modeling Language). We showed that this framework 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 a pivot language. With SmartSync, we studied the electrical power supply system. We checked consistency between system architecture and safety models. 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.

Some questions about the comparison of heterogeneous models remain open. As future works, we plan to improve the SmartSync platform, notably by developing new comparison algorithms and abstraction methods.

References

1. Abadi, M., Cardelli, L.: A Theory of Objects. Springer-Verlag, New-York, USA (1998)
2. Batteux, M., Prosvirnova, T., 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. Batteux, M., Prosvirnova, T., A.Rauzy: Altarica 3.0 in 10 modeling patterns. International Journal of Critical Computer-Based Systems (IJCCBS) **9**, 133 (2019). https://doi.org/10.1504/IJCCBS.2019.10020023
4. Batteux, M.B., Prosvirnova, T., Rauzy, A., Yang, L.: Reliability assessment of phased-mission systems with AltaRica 3.0. In: 3rd International Conference on System Reliability and Safety (ICSRS 2018). Barcelone, Spain (Nov 2018)
5. Bouissou, M., Bon, J.: A new formalism that combines advantages of fault-trees and Markov models: Boolean Logic Driven Markov Processes. Reliability Engineering and System Safety **82**, 149–163 (2003)
6. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixpoints. In: Proceedings of the 4^{th} ACM-Sigplan Symposium on Principles of Programming Languages, POPL'77. pp. 238–252. ACM, Los Angeles, California, USA (1977). https://doi.org/10.1145/512950.512973
7. Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann. The MK/OMG Press, San Francisco, CA 94104, USA (2011)
8. Fritzson, P.: Principles of ObjectOriented Modeling and Simulation with Modelica 3.3: A CyberPhysical Approach. Wiley-IEEE Press, Hoboken, NJ 07030-5774, USA (2015)
9. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y., Gottmann, S., Engel, T.: Model synchronization based on triple graph grammars: Correctness, completeness and invertibility. Softw. Syst. Model. **14**(1), 241–269 (Feb 2015). https://doi.org/10.1007/s10270-012-0309-1
10. Klee, H., Allen, R.: Simulation of Dynamic Systems with MATLAB and Simulink. CRC Press, Boca Raton, FL 33431, USA (February 2011)
11. Krob, D.: CESAM: CESAMES Systems Architecting Method: A Pocket Guide. CESAMES, http://www.cesames.net (January 2017)
12. Mauborgne, P., Deniaud, S., Levrat, E., Bonjour, E., Micaëlli, J.P., Loise, D.: Operational and system hazard analysis in a safe systems requirement engineering process application to automotive industry. Safety Science **87**, 256–268 (August 2016)
13. Noble, J., Taivalsaari, A., Moore, I.: Prototype-Based Programming: Concepts, Languages and Applications. Springer-Verlag, Berlin and Heidelberg, Germany (1999)
14. Pierce, B.C.: Basic Category Theory of Computer Scientists. Foundations of Computing, MIT Press, Cambridge, MA 02142-1315, USA (1991)
15. Rauzy, A., Haskins, C.: Foundations for model-based systems engineering and model-based safety assessment. Journal of Systems Engineering (2018). https://doi.org/10.1002/sys.21469
16. Roques, P.: MBSE with the ARCADIA Method and the Capella Tool. In: 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016). Toulouse, France (Jan 2016), https://hal.archives-ouvertes.fr/hal-01258014
17. Stephan, M., Cordy, J.R.: 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. pp. 265–277 (2013). https://doi.org/10.5220/0004311102650277
18. Yakymets, N., Julho, Y.M., Lanusse, A.: Sophia framework for model-based safety analysis. In: Actes du congrès Lambda-Mu 19 (actes électroniques). Institut pour la Maîtrise des Risques, Dijon, France (October 2014)