A Language-based Multi-view Approach for Combining Functional and Security Models

Hui Zhao*‡, Frédéric Mallet*†, Ludovic Apvrille§
Université Côte d'Azur, I3S, INRIA*
UMR 7271 CNRS†
Shanghai Trusted Industrial Control Platform Co., Ltd.‡
LTCI, Télécom Paris, Institut polytechnique de Paris§
{hui.zhao,frederic.mallet}@inria.fr, ludovic.apvrille@telecom-paris.fr

***Abstract*—The design flaws and attacks on Cyber-Physical Systems (CPSs) can lead to severe consequences. Thus, security and safety (S&S) issues should be taken into account with functional design as early as possible during the developing process. However, it's rare to see "one-size-fits-all" modeling language and/or design tool. One way to solve this issue is to integrate different nature models into one model system, but this requires a unified semantic among modeling languages. We explore a model-based approach for systems engineering that facilitates the composition of several heterogeneous artifacts (called views) into a sound and consistent system model. Rather than trying to extend either SysML or SysML-sec into more expressive languages to add the missing features, we extract proper subsets of both languages to build a view adequate for conducting a security and safety analysis of Capella (SysML-based) functional models. Our language is generic enough to extract proper subsets of languages and combine them to build views for different experts. Moreover, it maintains a global consistency between the different views.**

***Index Terms*—CPS, MDE, UML-like, SysML, SysML-Sec, ARCADIA, TTool, Multi-View Design, Security&Safety.**

I. Introduction

The design of Cyber-Physical Systems (CPS) demands to combine discrete models of pieces of software (cyber) components with continuous models of physical components [1]. Thus, the design of a CPS must involve several views of the system. Each view relies on specific expertise (i.e., mechanisms, aerodynamics, software, security, hardware, power, ...) and different Domain-Specific Modeling Languages (DSMLs). Different views are connected to each other by explicit associations that maintain global consistency. The diversity and heterogeneity of CPS make system design increasingly complex [2]–[4].

Model-Driven Engineering (MDE) is considered as a well-established software development approach that uses abstraction to bridge the gap between problem space and software implementation [5], [6]. MDE relies on models to describe complex systems at multiple levels of abstraction. In this paradigm, models are first-class elements that represent abstractions of a real system, capturing some of its essential properties. Models are instances of modeling languages which define their abstract syntax (e.g., using a metamodel expressed in a class diagram), concrete syntax (e.g., graphical or textual), and semantics (e.g., operational or denotational by means of a model transformation) [7]. As an essential issue of MDE, multi-view modeling integrates different models using various DSMLs (domain-specific modeling languages) and abstracting various aspects of systems and sub-systems, such as security&safety, and functionalities. Therefore, it is critical to understand the relationship between (meta) models.

In this sense, MDE is suitable for CPSs design as it can reduce the complexity of the systems and facilitate reasoning about functional and non-functional requirements [8], [9]. Specifically, MDE shortens development time, facilitates automated updates, mitigates flaws, and makes system models more natural to understand.

On one hand, The high complexity of systems requires a large domain of competencies, and experts in various domains have to work concurrently on different views/aspects of the same systems [10]. On the other hand, the security and safety issues take a vital role in the CPS, especially in some industrial critical system. In the paper, we thus propose a language-based approach for combining functional views with security and safety views, while each view remains exactly the same. New combined outcomes are exported as (meta) models. Despite different models of CPSs rely on different DSMLs and have specific natures, the proposed language can be used to describe the combined relationships among different models and to guide engineers to combine them. Finally, the combined models can be enriched by each other. The selected sub-models can be recognized by one of their support environments (tools) to perform verifications and/or simulations. In the end of this paper, a case study demonstrates how to use SysML-Sec [11] and its support environment TTool [12], [13] by combining UML-like models and SysML-Sec-based models. In this way, the security & safety properties are added to UML-like models. Then, They are able to perform security verifications and/or simulations.

Inspired by our previous work on combining Capella and AADL models [14] that effectively enriched Capella with AADL's capabilities so as to perform scheduling verification. This paper presents a safety and security oriented combination approach. The paper shows that our combination language can be effectively used to describe the combined relationships among (meta) models and guide engineers on how to combine functional models with security/safety models. Thus,

the combined models are capable of verifying security/safety properties.

The contributions of this paper are:

- A security-aware combined modeling approach.
- Propose a language for combining functional and security views at the metamodel level.
- Well-defined syntax and formal semantics for transformation rules.

The paper is structured as follows. The next section gives a brief background of Capella and TTool environments, while we present our motivations of this work. We introduce the model combination language in section III, the detailed definition of syntax and formal semantics are given as well. In section IV, we analyse the security and safety issues and related properties. In section V, we illustrate a case study about ADAS which demonstrates our approach and language are effective. In the end of this paper, we discuss related work and conclude this paper in Section VI.

II. Background and Motivation

A. Background

Recent years, the emerging ***Capella*** modeling solution [15] gives a comprehensive open-source framework for systems engineering design. It can be regarded as a subset of UML models and diagrams tailored for a given methodology, so-called *ARCADIA (ARChitecture Analysis and Design Integrated Approach)*. ARCADIA is a model-based engineering methodology for systems, hardware and software architectural design. It has been integrated in Capella project and developed by Thales since 2005 through an iterative process involving operational architects from all the Thales business domains (transportation, avionics, space, radar, etc.). It enforces an approach structured on successive engineering phases which establishes clear separation between needs (operational need analysis and system need analysis) and solutions (logical and physical architectures) [16]) in accordance with the ISO 42001 standard.

TTool is a SysML-Sec support toolkit [11], [13] which can capture system requirements, model software/hardware partitioning and model embedded software. Models of the different stages can be formally checked against security and safety properties. The different stages are linked by retro anotations that help confirming decisions taken during previous modeling stages.

TTool offers a press-button approach to verify models and to automatically retro-annotate models with verification results.

Relying on internal (simulation, model-checkers) and external tools (e.g., ProVerif and UPPAAL), TTool can perform simulation and formal verification for safety, security and performance [12]. Results can help engineer in deciding whether safety performance and security requirements are fulfilled [17], [18]. Especially, in TTool, model transformations translate the SysML models into an intermediate form that is sent into the underlying simulation and formal verification utilities. Backtracking to models is then performed to better inform the users about the verification results. Proofs of safety involve UPPAAL semantics [19], and security proofs use ProVerif [20].

B. Motivation

Security and safety issues are significant challenges for CPS design [21]. The risk of attacks and dangers on CPSs are growing rapidly, and they can lead to systems and property damage, even personal injury. Thus, security and safety issues should be taken into account as early as possible in system development processes. Thus, security vulnerabilities and safety flaws should be detected and mitigated. To this end, people must identify flaws and vulnerabilities as early as possible in CPS developement. Despite the large support offered by Capella, there is no direct native support for dealing with security and safety issues, while there are now several tools specifically tailored for security and safety, such as TTool. Since TTool is based on SysML-Sec, and Capella is also basically based on a UML profile, they have roughly the same core content and different specific features (see figure 1). The similarities between Capella and TTool (SysML-Sec) give us an intuitive conviction that there is a way to leverage TTool somehow to enrich Capella's security and safety analysis capabilities. The question that we address here is whether we can benefit from both Capella and security and safety tools without extending Capella. Extending Capella (integrating security and safety analysis capabilities into Capella) would make it more complex to learn and to use, and would raise maintenance and consistency issues. Rather than trying to extend Capella to adapt all necessary aspects, we propose to bring together small subsets of each of langages — Capella and SysML-Sec — to focus on specific analysis capabilities while keeping independence and global consistency of all these small pieces.

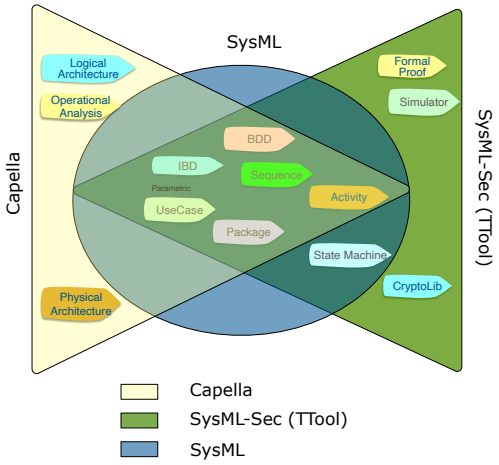

Fig. 1. Excerpt of relationships between SysML and SysML-Sec

In this sense, the proposed language is well suitable for building a set of relationships between Capella and other security/safety oriented tools at (meta) model level.

In this paper, we choose Capella as engineering modeling platform and TTool as security and safety analysis tool. Our

approach consists in extracting security and safety features by using metamodels at a high level and a set of operational methods. The former is an abstract representation of security that allows us to identify and verify security and safety properties formally, and the latter defines the operational process that is used to conducting transformation. A high-speed train controlling system serves as a use case which is used to demonstrate how engineering modeling design combines security and safety analysis with our proposed approach.

III. Model Combination Language

The proposed language is a dedicated (meta) language to extend and enrich one the capability of a DSML with the capability of another DSML. With our combination language, an integration engineer can explicitly capture combination scenarios at language level. Combination pattern can be used to specify different combination relationships. Specific operators are provided to build up Transformation Rule Expressions (TRE). A set of TRE defines a Transformation Rule Library (TRL) which specifies how to combine different (meta) models' elements. Once a TRL has been completed, it can be parsed by an automatic tool. Then, the tool can perform model transformation automatically. The concept of combination language is illustrated in figure 2.

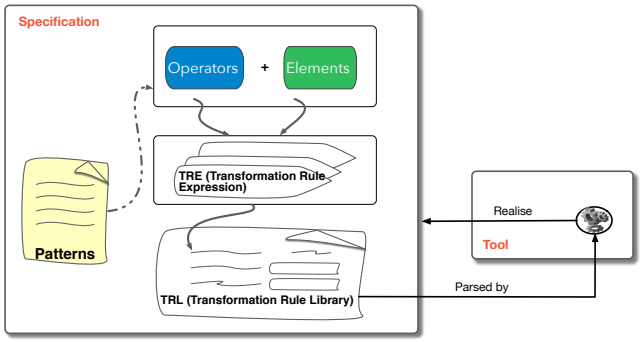

Fig. 2. Concept of Combination Language

A. Specification

A specification consists of combination patterns and corresponding TRL. It defines what and how elements from different models are combined. Once it is specified, integration experts can share this specification thus allowing the reuse and tuning of TRLs. As a specification can explicitly describe combination relationship, it can be also used to decompose models by bi-directional techniques for some decomposition needs.

B. Combination patterns

We have predefined several essential combination patterns, which provide all the declarations used in all the following examples. However, thanks to our language, designers can build other combination patterns depending on their problems and requirements. Of course, it is very likely that essential combination patterns must be extended for a given problem..

1) ***Association***: The association pattern is the most common and easy tool to understand since it is used to indicate that one element is associated to another element and their related sub-elements (for example, its embedded element or associated attributes).
2) ***Removal***: The removal pattern specifies situations where some elements are not considered for new models according to requirements.
3) ***Correspondence*** The Correspondence pattern captures an equivalence relationship among a set of elements.
4) ***Notation***: The notation pattern aims to hint people to add some extra information missing in the model. For example, the dependency relationship among the model's elements, and the nature of the elements.

C. Abstract syntax of Combination Language

The abstract syntax of our Combination Language is described with a metamodel expressed in a class diagram (shown in figure 3). The major element of Combination Language is a specification that contains Patterns, Operators and TRL. The specification requires importing at least two (meta) models. The imported (meta) models serve as a source of a set of candidate elements for following operations. An operator selects the elements and their attributes from imported (meta) models, and it also specifies how to combine selected elements with a clear relationship.

Each operator contains a Transformation Rule Expression which relies on a strict definition by EBNF (Extended Backus-Naur Form). Symbols are used to construct the TRE. For instance, for adding security properties to a logical component of Capella, the rule shall specify the corresponding element and their related attributes in SysML-Sec.

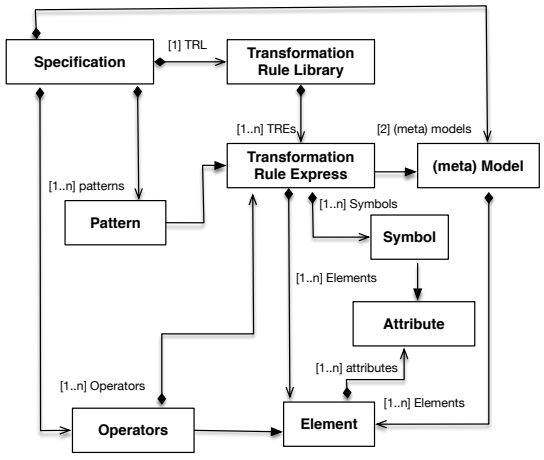

Fig. 3. A simplified view of abstract syntax of combination language

D. Meta symbol and notations rule expression

In this subsection, we firstly introduce some notations and meta symbols which are fundamental elements for constructing the well-defined Transformation Rule Expressions (see table I). In order to have non-ambiguous Transformation Rule

Expression patterns, we use EBNF to define TRE. EBNF is a notation technique for context-free grammars which often used to describe the syntax of languages [22].

TABLE I
SYMBOLS OF TRANSFORMATION RULE EXPRESSION

Symbol	Meaning
Γ	Beginning of transformation Rule
;	End of transformation rule
:	Separate elements
=>	Transforming
<>	Parent node
{ }	Attribute
[ ]	Optional value
|	Alternative
+	Object to be created
¬	Ignoring
@	Notation

The detail literal meaning of symbols are below:

1) A Transformation Rule Expression begins with "Γ" and ends with ";". The symbol "Γ" also can be considered as a boolean function, if $\underline{\Gamma(source,target)}$ is true, means there are interrelationships between *source* and *target*.
2) The symbol "=>" indicates a transforming action.
3) A transforming action contains the source elements which in the left side of "=>" and the target elements in the right side. A simple example is as bellow:

$$\Gamma < parent > source => target;$$

4) Symbol ":" separates each part of TRE. e.g, $P_{ort}\{Direction\} : \{Type\} : \{Secure\}$
5) An angle brackets "<>" encloses the parent node if the element has one or more parent nodes.
6) A parentheses "{ }" enclose attributes
7) A square braces "[ ]" delimit optional elements.
8) The alternative value is separated by a pipe "|". For example, The *port* has a directional attribute called *Direction* which could be *in* or *out* shown as:

$$P_{ort}\{Direction[in|out]\}$$

9) Symbol "@" indicates the notations which are used to add some extra informations such as dependency and nature. The extra informations are handled as the same as operational value: enclosed in [ ]; separated by ",". For example, $P_{ort}@[ModelA, Security]$ means element Port belongs to ModelA and is used for Security purpose (view). In such situation, it makes tools automatically display or hide the element P_{ort} which is in modelA and for security view in the following process.

A fewdetailed examples of Transformation Rule Expressions are shown in the listing 1.

Remarks:

- The symbol Γ in function expression does not have the same meaning as the meaning it has when put is at the beginning of transformation rules. To distinguish the function and the transformation rules, the formula is underlined in the following text.
- Relationships are defined in subsection III-F1

E. Abstract syntax of rule expression in EBNF

As we mentioned in the previous subsection, the TRE consists of one or more sequences of symbols. We list here the context-free syntax in EBNF in this subsection.

⟨*expression*⟩ ::= Γ ⟨*term*⟩ => ⟨*term*⟩;|
⟨*expression*⟩:⟨*term*⟩;|
⟨*operator*⟩ ⟨*term*⟩;

⟨*term*⟩ ::= ⟨*element*⟩|
⟨*operator*⟩⟨*element*⟩|
⟨*element*⟩⟨*operator*⟩⟨*element*⟩

⟨*operator*⟩ ::= '@' | '+' | '¬' |'=>'

⟨*element*⟩ ::= ⟨*element*⟩|⟨*attribute*⟩ |⟨*optional value*⟩

F. Operators and semantics

The context-sensitive syntax and the operational rules could also be considered as semantics instead of syntax. For example, the context-sensitive syntax is called static semantics in the UML specification documents from OMG [23]. In our case, it specifies how an instance of a construct can be meaningfully connected to other instances.

In order to define TREs is a more precise way, we formally define a set of relationships. Additionally, we propose a set of operators to build up TRE, which represents operations between (meta) models (e.g., transforming, creating, ignoring) in a systematic way. Both may also help users to understand the following TRE examples.

1) Definition of relationships: Here we define a set of essential relationships, which are used to describe the logical links between two model elements. Let A and B be sets of elements respectively, with a, b, c and x, y, z: elements of model, written as $A \supsetneq a,b,c$ and $B \supsetneq x,y,z$.

- **Relationship**: If the ordered pair (a,x) has a relationship, we written as $\mathcal{R}(a,x)$ or $a\mathcal{R}x$ for simplicity with $\mathcal{R}$ being a boolean relation function. "$\mathcal{R}(a,x)$ is true" means existing a relation between a and x, which implies that a can be transformed into x. It is written as:

$$\mathcal{R}(a,x) \implies \underline{\Gamma(a,x)}$$

  Obviously, the "**Relationship**" differs from its literal meaning. It is a function keyword used to indicate the existing of transformable relationship between elements.
- **Equivalence**: $\mathcal{E}(a,x)$ is a boolean function that is true if and only if a is semantically equivalent to x. If function $\mathcal{E}(a,x)$ is thrue, then

$$\mathcal{R}(a,x) \wedge \mathcal{E}(a,x) \implies \underline{\Gamma(a,x)}$$

  .
- **NotIn**: $\neg a$ is a boolean function. If it is true, then it means that there are no corresponding elements in set of x,y,z,

which neither have a relationship with a, nor semantically equivalent to a. Formally,

$$\neg\mathcal{R}(a,\{x,y,z\}) \vee \neg\mathcal{E}(a,\{x,y,z\}) \\ \implies \underline{\neg\Gamma(a,x),\neg\Gamma(a,y),\neg\Gamma(a,z)}$$

2) Definition of Operators:

(a) ***Transforming operator:*** $\implies$. For example, $a => x$ means that a transforms to x, if and only if $\mathcal{E}(a,x)$ is true, in other words, a and x have an *Equivalence* relationship.

(b) ***Creating operator:*** the name of an attribute within parentheses with plus "{ }+" indicates the creation of an element with the creation options following the "+". For example, $\Gamma a => x\{t\}+$, means that a transforms to x wit the addition of attribute t, if and only if $x \supsetneq t$ and $\mathcal{E}(a,x) \wedge \mathcal{R}(a,t) \wedge \mathcal{R}(x,t)$ is true.
For instance, in the following rule, the function port $Port_{fun}$ is transformed to a communicating port $Port_{comm}$, and a new attribute *Type* is created. The latter associates to communicating port $port_{comm}$ three optional values (data, event, data and event). :

$$\Gamma Port_{fun} => Port_{comm}\{Type[data|event|dataevent]\}+;$$

(c) ***Ignoring operator:*** used to ignore elements. It is denoted with symbol "¬" beforean element. For example, $\neg a$, it means a is **NotIn** object for a set B, in other words, we can neither find out ***Relationship*** nor ***Equivalence*** between a and B. Formally,

$$\neg\mathcal{R}(a,B) \vee \neg\mathcal{E}(a,B)$$

(d) ***Notation Operator:*** used to tag the nature of an attribute of an element e.g. to catalog an element fir displaying or selection.
For instance, $P_{ort}@[ModelA, Security]$ states gives two tags to P_{ort}. One tag is *ModelA*, indicating that the element P_{ort} belongs to *ModelA*. In other words, It represents a dependency relationship between this element P_{ort} and element *ModelA* or element *Security*.

G. Operational transformation rules

The following example better explains transformation rules (Please refer to the TRE table which is in the listing 1).

```
1 ΓPort{Direction[in|out]} => PrimitivePort{ Direction [ in|out]}:
  {Type[data|event|data event]}+;
2 Γ Function => <CompositeComponent>PrimitiveComponent{
    AccessIdentifier}+:{InitialValue}+:{Type[natrul|boolean]}+;
3 Γ¬Port{ordering};
4 ΓExfun{Source} => <connections>:connection:{source};
5 ΓExfun:{Target} => <connections>:connection:{target};
```

Listing 1. The example of transformation Rule Expressions

In line 1, the rule transforms an element *port* (it has direction attribute) of source model to an target object element *port*, adding a new attribute *Type* with three optional value (date, event or data event). These "type value" qre expected to be known by target model's DSML and the their support tool. The added attribute can be used to continue further design. In line 2, it is similar to previous one, but the object element *function* has a parent node called *CompositeCpmponent* which is enclosed in a pair of angle brackets.

Then, in line 3, an element of the source model is ignored because it has no correspondence in the target model, or the source element is not needed by the target model.

Finally, in line 4 and 5, a *Equivalence* relationship between the source elements and the target elements gives a one by one transformation e.g. "Source" to "source" and "Target" to "target", respectively.

IV. Multi-view modeling approach for security/safety design

In the industrial field, domain experts usually design several (meta) models (e.g. functional and architectural) to deal with the requirements of stakeholders at the design stage. Traditional functional models describe only two types of functions: physical and cyber. Functions interact with each other through flows that may reflect the non-directional exchange of energy, data or signal flows. These flows carry real physical and cyber properties such as mechanical, electrical, thermal energy, and data. Thus, existing functional models naturally leak information that can be used to attack the system via the signal flows in the cyber domain or energy/material flows in the physical domain. We extend the functional modeling concept and combine different domain (meta) models by using proposed language which is presented in the above sections. In this section, we introduce a security/safety-oriented multi-view modeling approach, with the objective to analyze the cyber security of Capella artifacts, as well as the possible countermeasures and their impact on the performance of the system, with TTool as the underlying proof framework.

A. Security and Safety scopes

Security Requirements describe which system elements must not be recoverable or modifiable by the attacker. Also, the system must deny unidentified access or unknown modified command/data. Typical security requirements refer to *Confidentiality, Authenticity/Integrity, Availability, and Non-Repudiation*.

Safety refers e.g. to the inability for a system to reach deadlocks/livelocks, error states, and delayed reactions to safety-critical events. Ensuring the safety of users or bystanders involves considering multiple factors. Conventional safety suggests that a system should not contain software and hardware flaws which can prevent it from correct function. "Safety of the Intended Function" involves avoiding the situations where the system or its components cannot handle, such as adverse extreme environmental conditions. *Timing* can be critical for certain real-time systems as it will need to respond to certain events as quickly as possible, such as obstacle avoidance and reducing speed, within a set period to avoid dangerous situations. Any extra delay (with regards to deadlines) may results in a quite severe consequence.

B. Properties to Verify

To ensure that the system works as designed, safety and security verifications are useful means, and they have to be performed. There are some safety properties that can be checked through safety verification, such as reachability, liveness. These properties can be formalized and checked by the model-checker such as UPPAAL or with reachability graphs [24]. TTool is able to transfer those formal properties to model-checker and get results from model-checker to notify users [25]. As for security properties, such as authenticity, confidentiality and Access Control etc. We also can rely on TTool to verify these security properties. This is also the reason that we chose TTool to complete Capella's capability.

We give some definitions of safety and security properties as below:

- **Reachability** is a property that can determine if a function or condition is present in at least one execution path of the system. It also can indicate if the model is correct and all the functions and conditions can be executed as good as designed.
- **Liveness** is a property that can not be violated in a finite execution of an embedded system. For example, after one event occurs, another event will always be trigged. A good event should only occur at some time after execution ends.
- **Confidentiality** is a property that indicates whether an attacker or unauthorized person can get and read the communicating data in the system. If the attacker can only get the data, but can not read it. The communicating data is proved confidential, and it also proves that the encryption algorithms are effective.
- **Authenticity** is a property that indicates whether communicating data within a system can be modified by an attacker or an unauthorized person.
- **Availability** is a metric that measures the system's usability. In other words, it indicates if a system is not failed or on a repairing status when it needs to be used. For example, if the system can provide services immediately when requested by authorized users.
- **Access Control** is a security technique that allows only authorized entities to use resources or perform specific actions in the computing system. It can be related to both Confidentiality and Authenticity [24], as an unauthorized entity is not able to access confidential data, and should not be able to modify any code of a system and invoke any internal components of the system. Access control techniques should prevent insecurity actions and deny unauthorized service requests. It is a fundamental concept in security that minimizes risk to the system.

C. Transformation rule Library for security & safety

By using the proposed combination language, we can construct a set of relationships between functional meta-models and security/safety-oriented meta-models. The set of relationships is called TRL, which we mentioned in the above

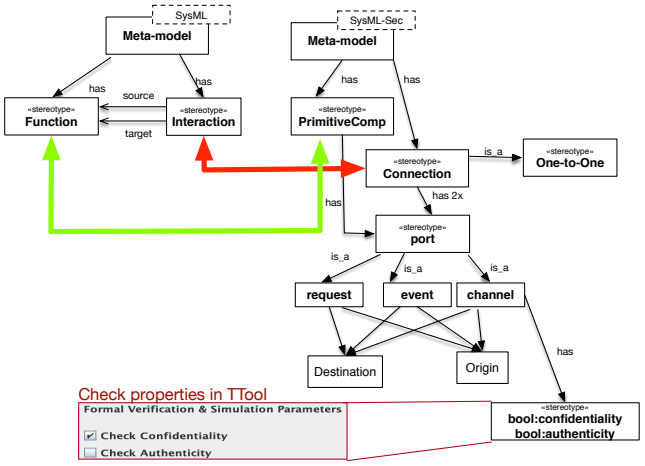

Fig. 4. A simple schema of relationships between security meta-model of TTool and functional meta-model of Arcadia

sections. Once the TRL is established, the following process of generating could be automatic by the tool. As the combined models include both the functional and security parts, we can import those models to TTool for security/safety analysis (simulation, verification). The results can be traced back to the functional design part. A simplified schema of relationships between Arcadia meta-model and TTool meta-model is shown in figure 4. The green flash links to two functional elements. They are the equivalence relationship. The red flash links two security-related elements, because "connection" and its sub-element "port" have some attribute which can associate some encryption techniques (i.e., AES). In the red box on the left bottom of the figure, there are two parameters in TTool for formal verification and simulation. They associate with the channel component in TTool, and this "channel" can be linked to "interconnection" in Capella (SysML) and added new properties. If the "confidentiality" is checked, the corresponding algorithm will be applied to encrypt the data which is sent by this channel. As to "Authenticity", it's the same as "confidentiality". In other words, the functional components in Capella can be seen as they have additional security and safety properties as long as they are linked to a TTool (SysML-Sec) component using our proposed language.

1) An instance of TRL: TRL serves transforming meta-model of Capella into a new meta-model, which is added a part of TTool's meta-model.

Duo to space limitations, the detailed TRL is not presented here, as most of them are written in the same way as the following listing of a piece of TRL 2:

```
1 Γ Function => PrimitiveComp;
2 Γ Interaction => Connection;
3 Γ Port => <Connection>Port{Type[request|event|channel]}+
   : Port{Origin[Destination|Origin]}+;
4 Γ Port => <Port>channel{confidentiality[true|false]}+
   :<Port>channel{ authenticity [true|false]}+;
5 ...
```

Listing 2. An example of Transformation Rule Library for security purpose

V. CASE STUDY

ADAS (Advanced driver-assistance systems) are typical CPSs and play an important role for autonomous vehicle. Conventional ADAS technology can detect some obstacle, alert the driver of hazardous road conditions, in some cases, slow or stop the vehicle. This level of ADAS is great for applications like blind-spot monitoring, lane-centering assistance, obstacle avoiding, and forward collision warning. It means that the "driver is disengaged from physically operating the vehicle by having driver's hands off the steering wheel and foot off the pedal at the same time". However, the liberating driver also brings great risks, e.g, the underlying flaws are used by attackers to hijack the vehicle such as getting the remote control, delaying system response time.

In this case study, we demonstrate how to add safety and security verification abilities for Capella's functional design by using proposed approach. SysML-Sec further adds the safety and security properties for functional design. Then, we can perform verification to check if security and safety properties are satisfied. All the results get back to Capella to correct or adjust functional design.

We start with meta-models combination at meta-model phase (as shown in Figure 5 on left). Using our proposed language to build up TRL, which is presented in IV-C. Once the TRL has been done, we enter model phase for functional design on Capella (shown in middle of figure). All of the sensors (radar, camera, etc) and ADAS's control system tasks (Perception and Navigation) are designed as functions on the Capella, while model all the function exchanges.

Next, leveraging TRL, we can transform Capella models into SysML-Sec models for further safety & security design and analysis. All the required attributes and properties would be filled in TTool/SysML-Sec such as port's properties (direction, type etc). For example, according to the TRL (see listing 2), firstly, we write a TRE is "Γ Function => PrimitiveComp", while all the component which is classified as *function* will be found in Capella's model, and then, they are transformed to *PrimitiveComp* in TTool's model with their name. Secondly, the next TRE is "Γ Interaction => Connection", as the first one, all the *Interaction* components will be transformed to *Connection* in TTool. Next, we transform *Port* to *Port*, and add two new attributes *Type and Origin* for *Port* in SysML-Sec, according to the TRE "Γ Port => <Connection>PortType[*request*|*event*|*channel*]+: PortOrigin[*Destination*|*Origin*]+". The new concrete TTool model are generated once those rules of transformation have been successfully traversed. To do this automatically, we are developing a model combination tool[1] which is able to transform the components according to TRL.

For security verification, we have to set up the security properties in TTool such as cryptographic configuration, and further design associated activities. TTool then performs verification automatically, and gives a feedback as long as the the verification process is finished. Similarly, safety properties can be verified from the same SysML-Sec model: *Reachability*, *Liveness*, and absence of deadlocks.

[1]Ongoing development tool, refer to https://github.com/conanbos/MTool

VI. RELATED WORK

Multi-View approach allows to develop both software and hardware from different domains by quickly and effectively integrating different domain expert artefacts to build up a sound and consistent system. A lot of work is devoted to providing efficient dedicated (meta) language for integrating issues. For instance, Muller *et al* [26] proposed to use aspect-oriented modeling to build an executable meta-language by composing action metamodels, and Jézéquel worked at model weaving approach [27]. In contrast to their languages or approaches, our approach is dedicated to seamlessly combine different models of views at high-level, it is easier to use and understand. Other approaches addressed modeling consistencies from constraint-based [28] or from architecture models [29]. On our side, we tackle this problem with an efficient yet simple combination of (meta) models.

Jörg Kienzle *et al* proposed a composition technique which was implemented in a tool called Kompose [30]. Kompose focused mainly on the merging of class diagrams. In their proposition, the model elements to be composed must be of the same syntactic type, that is, they must be instances of the same meta model class.

Degueule *et al* [31] also provided a so-called "Melange" meta-language. This language can weave two DSLs to produce new DSLs that integrated the syntax and semantics of the two languages. Our approach is not to get a new language, but to take advantage of other tools to complete our needs by combining (meta) models.

In Thramboulidis *et al* [32] paper, they introduced a UML-based approach adapted to Internet of Things (IoT), so-called *uml4Iot*, which can automatically generate the process which is required for cyber-physical component to be integrated into the manufacturing environment. Our approach relies on well-defined language and focus more on security/safety aspects of embedded systems (including industrial control system, IoT and smart manufacturing, etc). Based on the metamodel combination method, it makes our approach more flexible.

VII. CONCLUSION AND DISCUSSION

In this paper, we proposed a language-based multi-view approach for combing functional and safety&security models of CPSs. The syntax of the language is well-defined in EBNF, formal semantics is given as well. The proposed language is used to describe the combined relationships among models, which have specific natures and rely on different DSMLs. By using this, system-level modeling and verification of safety&security are enabled during the concept design stage of CPSs. With the proposed approach, combined models can be enriched and leveraged by each other so as to perform combinational verification and/or simulations. A safety&security-aware design case of autonomous vehicle system has demonstrated how the functional models equip safety&security capabilities by using

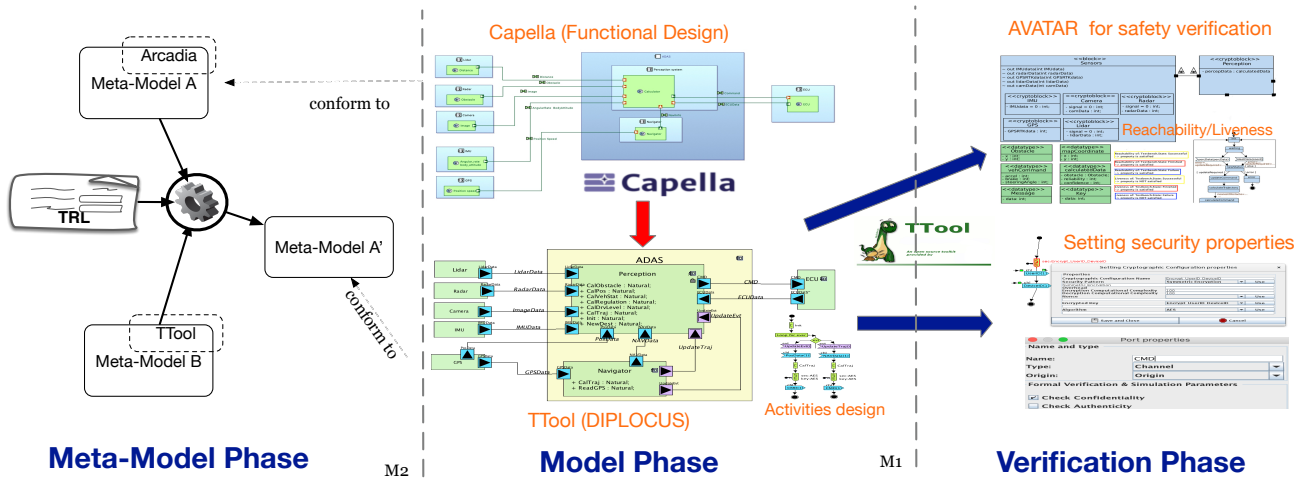

Fig. 5. Work flow of ADAS design from meta-model combination phase to Verification phase for security and safety purposes

the proposed language, and performing combinational safety& security verifications and/or simulations by TTool toolchain.

REFERENCES

[1] E. A. Lee, "Cyber physical systems: Design challenges," in *11th IEEE ISORC*, May 2008, pp. 363–369.

[2] D. Garlan, "Modeling challenges for CPS systems," in *IEEE/ACM International Workshop on SEsCPS*, May 2015, pp. 1–12.

[3] B. Radhakisan and H. Gill, "Cyber-physical systems," *The Impact of Control Technology*, vol. 12, no. 1, pp. 161–166, Mar. 2011.

[4] R. Rajkumar, L. Insup, S. Lui, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Design Automation Conference*, Jun. 2010, pp. 731–736.

[5] H. Ergin, E. Syriani, and J. Gray, "Design pattern oriented development of model transformations," *Computer Languages, Systems & Structures*, vol. 46, pp. 106–139, Nov. 2016.

[6] T. Stahl, M. Voelter, and K. Czarnecki, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc., 2006.

[7] D. Harel and B. Rumpe, "Modeling languages: Syntax, semantics and all that stuff," Tech. Rep., Aug. 2000.

[8] E. A. Le, "Cyber-physical systems-are computing foundations adequate," *NSF workshop on cyber-physical systems: research motivation, techniques and roadmap*, vol. 2, pp. 1–9, Oct. 2006.

[9] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, Jan. 2011.

[10] C. Gomez, J. DeAntoni, and F. Mallet, "Multi-view power modeling based on UML, MARTE and SysML." *38th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 17–20, Sept. 2012.

[11] L. Apvrille and Y. Roudier., "SysML-sec: A sysML environment for the design and development of secure embedded systems," *APCOSEC*, pp. 8–11, Sept. 2013.

[12] L. Apvrille, L. Li, and Y. Roudier, "Model-driven engineering for designing safe and secure embedded systems," in *ACVI*. IEEE, Apr. 2016, pp. 4–7.

[13] Y. Roudier and L. Apvrille, "SysML-Sec: A model driven approach for designing safe and secure systems," in *MODELSWARD*, Feb. 2015, pp. 655–664.

[14] H. Zhao, L. Apvrille, and F. Mallet, "Meta-models combination for reusing verification techniques," in *7th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS-Science and Technology Publications, 2019, pp. 39–50.

[15] P. Roques, "MBSE with the ARCADIA Method and the Capella Tool," in *8th ERTS*, 2016.

[16] Capella, "Website of capella/arcadia," [Online]. Available: https://www.polarsys.org/capella/arcadia.html, 2019.

[17] L. Li, L. Apvrille, and D. Genius, "Virtual Prototyping of Automotive Systems: Towards Multi-level Design Space Exploration," *Conference on Design & Architectures for Signal & Image Processing (DASIP'2016)*, 2016.

[18] D. Genius, L. Li, and L. Apvrille, "Model-driven performance evaluation and formal verification for multi-level embedded system design," in *5th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS, 2017, pp. 78–89.

[19] D. Alexandre, K. G. Larsen, A. Legay, M. Mikučionis, and D. Bøgsted, "Uppaal SMC tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015.

[20] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "Proverif 2.00: Automatic cryptographic protocol verifier," May 2018.

[21] J. Wan, A. Canedo, and M. A. Al Faruque, "Security-aware functional modeling of cyber-physical systems," Sept. 2015, pp. 1–4.

[22] D. D. McCracken and E. D. Reilly, "Backus-naur form (BNF)," pp. 129–131, 2003.

[23] OMG, "Unified modeling language," [Online]. Available: https://www.omg.org/spec/UML/About-UML/, Apr. 2019.

[24] L. Li, "Safe and secure model-driven design for embedded systems," Ph.D. dissertation, Université Paris-Saclay, Sep. 2018.

[25] F. Lugou, L. W. Li, L. Apvrille, and R. Ameur-Boulifa, "Sysml models and model transformation for security," in *4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE, 2016, pp. 331–338.

[26] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel, "Weaving executability into object-oriented meta-languages," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, Oct. 2005, pp. 264–278.

[27] J.-M. Jezequel, "Model driven design and aspect weaving," *Software and Systems Modeling*, vol. 7, no. 2, pp. 209–218, Feb. 2008.

[28] G. Yang, X. Zhou, and Y. Lian, "Constraint-based consistency checking for multi-view models of cyber-physical system," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Jul. 2017, pp. 370–376.

[29] A. Bhave, B. H. Krogh, D. Garlan, and B. Schmerl, "View consistency in architectures for cyber-physical systems," in *2011 IEEE/ACM Conference on Cyber-Physical Systems*, Apr. 2011, pp. 151–160.

[30] J. Kienzle, W. Al Abed, and J. Klein, "Aspect-oriented multi-view modeling," in *8th ACM Conference on Aspect-oriented Software Development*, 2009, pp. 87–98.

[31] D. Thomas, B. Combemale, A. Blouin, O. Barais, and J.-M. Jezequel, "Melange: A meta-language for modular and reusable development of dsls," in *2015 ACM SIGPLAN Conference on Software Language Engineering*, 2015, pp. 25–36.

[32] K. Thramboulidis and F. Christoulakis, "UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems," *Computers in Industry*, vol. 82, pp. 259–272, Oct. 2016.