Meta-models Combination for Reusing Verification Techniques*

Hui Zhao[1], Ludovic Apvrille[2] and Frédéric Mallet[1]
[1]*Université Côte d'Azur, I3S, INRIA*
[2]*LTCI, Telecom ParisTech, Université Paris Saclay*
{*hui.zhao, frederic.mallet*} *AT inria.fr, ludovic.apvrille AT telecom-paristech.fr*

Keywords: CPSs, MBSE, AADL, Arcadia, Model Transformation

Abstract: The complexity of Cyber-Physical Systems (CPSs) is rapidly increasing because more and more aspects have been considered during the design phase. Each aspect involves its domain-specific modeling language (DSML), How to combine various DSMLs was a challenging problem. Rather than build-in all analysis and modeling capacities, we prefer with independent domain specific metamodels and link them together. In this paper, we show how to use a coordinated metamodel approach as a systematic way to gather diverse domain models and cross-cutting concerns. Thus, the paper proposes a set of transformation operators to manipulate (AADL and SysML) metamodels. We show that we can thus enrich platform capacities by blending different languages seamlessly, as well as perform the functional and scheduling analysis respectively through concrete models. A train traction controlling system serves as a case study.

1 Introduction

Developers of Cyber-Physical Systems (CPSs) have to deal with different domains, each of them having different characteristics. It is seldom the case that one development platform or a single language can adapt to all aspects with assumption one-size-fits-all. Therefore developers have to rely on domain-specific languages to handle the different domains problem. This results not only in a proliferation of languages but also increases the design complexity of CPS. Meanwhile, the gaps between languages and platforms brought additional problems such as coherency and consistency problems, which are exposed at integration and simulation stages. This also further exacerbates the complexity, make the complexity sky-rocketing.

To tackle these problems, engineers need an approach which can efficiently combine already existing languages. The goal is not only to combine the different languages seamlessly but also to benefit from the advantages of each language. Two solutions are possible. (i) A first solutions is to continuously integrate the necessary languages into an existing development platform, thus leading to progressively build a comprehensive development platform. However, this could lead to a never ending process resulting in a gigantic framework, which would be difficult to use, maintain, etc. (ii) A second solution is keeping each language (or tool) isolated, and relate some of the elements of each language (sub) meta-model in order to conduct the different analysis offered by each approach (e.g. scheduling analysis, safety analysis, etc.). In this second solution, each domain expert can work independently. Yet, since each language has its own characteristic such as syntax and semantics, we have to eliminate the gaps between them and handle the consistency issues. Therefore, our contribution is to propose a formal combining approach to link two modeling languages. To do so, we define how to relate two (sub-)metamodels. Then, once two models m_1, m_2 has been performed in the two languages, m_2 can automatically be augmented with some of the information of m_1 in order to perform verification on enriched models (e.g., scheduling, timing, safety), and then to backtrace the verification results to m_1.

In this paper, we selected SysML and AADL to demonstrate the relevance of our approach. These languages are supported by the tools Capella/Arcadia and OSATE2 [2], respectively. The paper is organized as follows. We present our contribution with first our workflow and then with the formal definition of a set of combination rules and operators. Then, in section 3, we apply these operators on functional

*This work was financially Supported by the CLARITY project and by a UCN@Sophia Labex scholarship.

[2]http://osate.org/index.html

and physical views. In section 4, train traction controlling systems are used to demonstrate architecture and scheduling analysis. Section 5 presents the related work. Finally, section 6 concludes the paper and gives our future work.

Some special notes for this paper are: 1) When we mentioned Arcadia, it means SysML-based methodology, the language is SysML. 2) In sections 2, 3 and 4, all elements on the left of transformation rules belong to metamodels of Arcadia and all elements on the right are from the AADL metamodels. The two metamodels have been imported by default, and we thus omitted the prefix (e.g., *MM.Arcadia*.function) for pithiness.

2 Approach

2.1 Workflow

The workflow of our approach is shown in figure 1. While Arcadia is well adapted to describe allocations of functions, AADL rather focuses on the concrete execution behaviors of components. In this paper, we use transformation to enhance Arcadia with the scheduling analysis features of AADL. To do so, we propose a set of rules and operators to specify the relationships at the M2 level. These rules — defined in a Transformation Rule Library or TRL for short — an establish a relationship between Arcadia and AADL metamodels. For instance, Arcadia and AADL define concepts that can be put in relation with our rules. For example, an Arcadia function allocated to a processor can be related to a "thread" in AADL, as shown as green part in the figure. When a feature has no equivalence in Arcadia, additional attributes must be added (e.g., period and execution time, shown as red part). We manually choice the elements of metamodel depend on requirements of the project. Once the equivalence relations between the two metamodels have been settled, we can get a temporary combinational metamodel (TCM) at run time by using TRL (step 1). Then, the TCM can be used to enhance an AADL model with elements of an Arcadia model (step 2). Then, the new AADL model can be exported into OSATE for further editing. Next, it can be used to perform the scheduling simulation in the Cheddar (Singhoff et al., 2004) analysis tool (step 3). This tool can be used to detect design flaws, conflicts of time and resources. We then expect to trace back the results to the Arcadia model in order to help the designer enhancing the performance of his/her model (step 4).

2.2 Operators and Rule expression

Metamodels can be manipulated with formal writing rules built upon operators e.g. transform, create, ignore. In the following parts of the paper, several operators and their semantics are defined (see table 1).

Symbol of operator	Meaning
Γ	Transformation Rule
;	End of rule
:	Separate elements
$\rightsquigarrow$	Transfer
<>	Parent node
{ }	Attribute
[ ]	Optional element
|	Separation of elements
{ }+	Attribute to be created
¬	Ignore

Table 1: Symbols of transformation rule expression

2.2.1 Structure of rule

A rule begins with "Γ" and ends with ";". A transfer symbol $\rightsquigarrow$ contains a source object in its left side and one or more target objects in its right side. The parent node is enclosed within the angle brackets "<>" (if it has one parent node). Finally, a rule is as below:

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

Each part of an object is separated by ":". The attribute group is enclosed with parentheses "{ }". square braces "[ ]" delimit optional elements and the alternatives are separated by a pipe "|". For example, The port has a directional attribute called Direction which could be in or out. Shown as below:

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

2.2.2 Creating operator

In the case of creating a new attribute, put the name of an attribute in the parentheses with plus "{ }+", is used to present the option is to be created. An example as below, an attribute type of component port will be created with three optional values (data, event, data and event):

$$Port : \{Type[data|event|dataevent]\}+$$

2.2.3 Ignoring operator

For some ignored attributes and objects are denoted with "¬" which is in front of the ignored object.

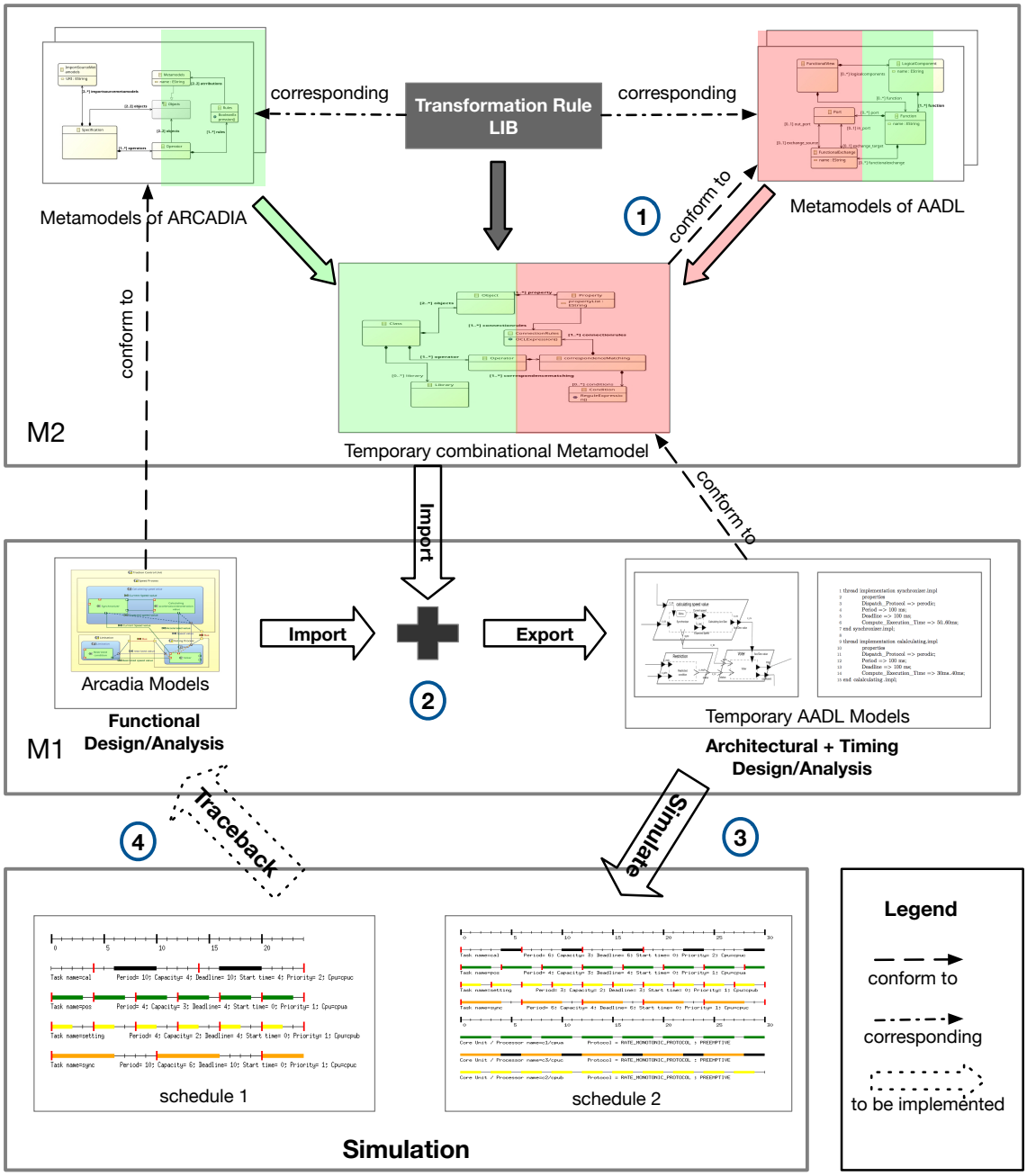


Figure 1: Overview of Workflow

An example of transformation rule expressions is in listing 1. In particular, the number of attributes of the target object may be greater than the number of attributes of the source object in the case of a new object created.

columns

```
1 ΓP_ort:{Direction[in|out]}⇝ <feature>:
      Port:{Direction[in|out]}:{Type[data|
      event|data event]}+;
```

```
2 ΓPP ⇝ <feature>:Port:{Direction[in|out
      ]}+:{Type[data|event|data event]}+;
3 ΓP_ort:¬{ordering});
4 ΓEx_fun:{Source}:{Target} ⇝ <connections
      >:connection:{source}:{target};
```

Listing 1: An example of transformation rules

2.3 Arcadia2AADL model transformation

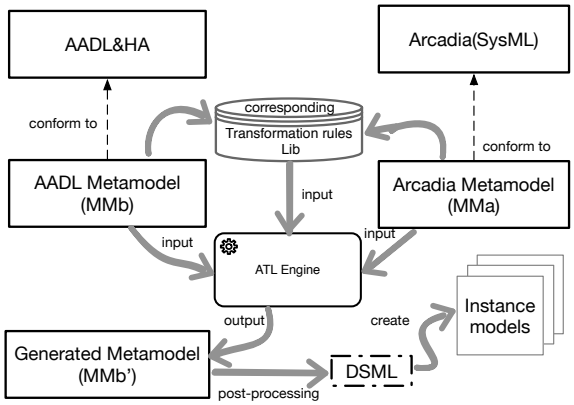


Figure 2: Structure of Arcadia2AADL model transformation

Based on the defined transformation rules, The Arcadia2AADL transformation can create a temporary metamodel according to original metamodels and TRL. The ATL serves as a transformation engine. For more detailed information of ATL, the reader is referred to (Jouault et al., 2006)(Jouault et al., 2008). The Structure of Arcadia2AADL model transformation is shown in Fig 2. AADL metamodels (denoted as MMb) and Arcadia metamodels (denoted as MMa) are imported (input) into ATL engine, and the engine is going to read transformation rules library and then the transformation engine exports (output) a temporary metamodel (denoted as metamodel MMb') which is a union subset of AADL and Arcadia and conform to AADL metamodel. The generated metamodel is then further post-processed as a DSML (domain-specific modeling language) in *ecore* format which can be used to create instance models. Therefore, all instance models afterward are conformed to this metamodel and will be used for further simulation and analysis.

3 Transformation Rule Library

As we described in the above section, the Transformation Rule Library (TRL) play an important role in the transformation process. Hence, in this section, we present how to construct a TRL from the following three views, functional view and physical view. Each view contains the metamodels which are from the subset of AADL and Arcadia.

3.1 Functional view

3.1.1 Logical components in Arcadia

The logical components in Arcadia contain a set of member elements, such as logical component containers, functions, ports, and functional exchanges. In the Arcadia, Functional diagrams consist of a set of SysML blocks and its interactions, named *Logical components*; The notion of Logical components enables better expression of system engineering semantics compared to SysML, and particularly, reduces the bias towards software. SysML block definition diagrams (BDDs) and internal block diagrams (IBDs) are assigned to different abstract and refined layers, respectively. The definition of a block in SysML can be further detailed by specifying its parts; ports, specifying its interaction points; and connectors, specifying the connections among its parts and ports. This information can also be visualized using logical components in Arcadia. In the definition 3.1.1, we present a metamodel of an instance of logical components.

Definition 3.1.1. (Logical Component) A logical component (**LC**) is 5 tuples,

$$\mathcal{LC} = < C_{omp}, F_{un}, P_{ort}, Ex_{fun}, M_{cf} >$$

where, $C_{omp} = \{\bigcup_{F_{un}}\}$ is a logical component container which contains a set of functional elements. F_{un} is a finite set of functional block include their name and id attributes. P_{ort} is a finite set of functional ports including directions and allocation attributes. $Ex_{fun} \subseteq P_{ort} \times P_{ort}$ denotes a finite set of functional exchange (connection) between two functional ports, it must be pair, one is source, another is target. M_{cf} : $\Sigma F_{un} \rightarrow C_{omp}$ allocate functions to a logical component container.

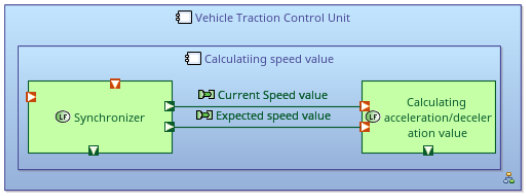


Figure 3: An example of functional view of vehicle traction control unit in ARCADIA

In the figure 3, there is a functional instance model of a part of a vehicle traction control unit in ARCADIA as an example. The blue rectangle is named logical component in Arcadia, but we consider it as a function's container, we thus call it *logical component container* C_{omp} in this paper. The green rectangle are functions F_{un} which are contained by C_{omp}. The element M_{cf} has represented this

allocation relationship between logical component containers and functions $M_{cf} : \Sigma F_{un} \rightarrow C_{omp}$. The deep green square with the white triangle is the outgoing port (P_{ort}), which connects to an incoming port (P_{ort}) that is drawn as a red square with white triangle and the green line is the functional exchange between two functional ports (Ex_{fun}).

3.1.2 The metamodels of software in AADL

AADL is able to model a real-time system as a hierarchy of software components, predefined software component types in the category of the components such as thread, thread group, process, data, and subprogram are used to model the software architecture of the system.

Definition 3.1.2. (Software Composition) A **SC** is a 4-tuples:

$$\mathcal{SC} = < Type, Port, Connection, Annex >$$

where *Type* specifies the type of components (e.g, system, process, thread). *Port* is a set of communication point of component. Port could be different types such as **data** port, **event** port and **data event** port. And, port can specify the direction such as **in** port, **out** port, **in out** port. *Connection* is used to connect ports in the direction of data/control flow in uni- or bi-directional. *Annex* is defined for the refinement of component, in this paper, we used hybrid annex to explicitly describe the both discrete and continuous behavoir of train traction control system.

3.1.3 Hybrid Annex

We use the HA to declare both discrete and continuous variables in the *Variables* section, and the initial values of constants are given in *constant* section. *Assert* is used to declaring predicates which may be used with invariants to define a condition of operation. The *behavior* section is used to specify the continuous behavior of the annotated AADL component in terms of concurrently executing processes, and use continuous evolution — a differential expression to specify the behavior of a physical controlled variable of a hybrid system. The communication between computing units and physical components are an essential part of a hybrid system, Communication between physical processes uses the channels declared in the *channel* section, and communicate with an AADL component relies on ports that are declared in the component's type. Continuous process evolution may be terminated after a specific time or on a communication event. There are invoked through timed and communication interrupt, respectively. A timed interrupt preempts continuous evolution after a given amount of time. A communication interrupt preempts continuous evolution whenever communication takes places along any one of the named ports or channels. The definition 3.1.3 gives a metamodel of Hybrid Annex which does not exist in SysML-based environment.

Definition 3.1.3. (Hybrid Annex) A *Hybrid Annex* is a 8-tuples:

$$\mathcal{HA} = < Ass, Ivar, Var_{hd}, Cons_{hd}, P_{roc}, ChP, Itr, B_{itr} >$$

where *Ass* is a finite set of assert for declaring predicates applicable to the intended continuous behavior of the annotated AADL component. *Ivar* is associated with assert to define a condition of operation that must be true during the lifetime. Var_{hd} is a finite set of discrete and continuous variables. $Cons_{hd}$ is a finite set of constants which must be initiated at declaration. P_{roc} is a finite set of processes that are used to specify continuous behaviors of AADL components. *ChP* is a finite set of channels and ports for synchronizing processes. *Itr* is a finite set of time or communication interrupts. $B_{itr} : Itr \rightarrow P_{roc}$ binds interrupts to related processes.

3.1.4 Functional elements transformation rules

The table 2 shows the correspondence between AADL and Arcade elements. The Additional attributes column are the attributes to be created during the transformation. According to this table, we can easily write the transformation rules to transforming Arcadia to AADL on functional parts, denoted $\mathcal{LC} \rightsquigarrow \mathcal{SC} + \mathcal{HA}$. An example as below (listing 2):

columns

```
1 ΓC_omp ⇝ Type[ system|process ]:{
      Runtime_Protection[true|false]}+;
2 ΓF_un ⇝ Type[abstract|thread]:{ Dispatch\
      _Protocol[Periodic|Aperiodic|
      Sporadic|Background|Timed|Hybrid]}+;
3 ...
```

Listing 2: Functional elements transformation rules example

3.2 Physical view

3.2.1 Execution Platform in AADL

Processor, memory, device, and bus components are the execution platform components for modeling the hardware part of the system. Ports and port connections are provided to model the exchange of data and event among components. Functional and non-functional properties like scheduling protocol and execution time of the thread can be specified in components and their interactions.

Arcadia	AADL	Additional attributes
Logical component container (C_{omp})	System,Process	{Runtime_Protection[true|false]}+
Function (F_{un})	Abstract, Thread	{Dispatch_Protocol[Periodic|Aperiodic|Sporadic|Background|Timed|Hybrid]}+
Port (P_{ort})	Port	{Type[data|event|data event]}+
Functional Exchange (Ex_{fun})	Connection	◯
◯	Annex	{Type[abstract| thread]}:{annex}+
Physical Node (N_{ode})	Device,Memory,Processor,Bus	{Dispatch_Protocol}+:{Period}:{Deadline}+:{priority}+
Physical Port (PP)	◯	$\neg PP$
Physical Link (PL)	Bus/BusAccess	{Allowed_Connnection_Type}+:{Allowed_Message_Size}+: {Allowed_Physical_Access}+:{Transmission_Time}+

Table 2: Functional and Physical elements correspondence table

Definition 3.2.1. (Execution Platform) A **EP** component is defined as a 3-tuples:

$$\mathcal{EP} =< EC, BA, C_{onn} >$$

where,EC defines the execution component such as processor, memory, bus and device. BA defines the BusAccess which is interactive approach between **bus** component and other execution platform components. $C_{onn} \subseteq EC \times EC$ denotes a finite set of connection between two components via bus device.

3.2.2 Physical components in Arcadia

The physical component in Arcadia consists of physical Node, Port and Link. The Physical Port and Link correspond to port and bus connection in AADL. There are some choices when the physical Node is translated to AADL such as device, memory, and processor, hence the designer has to point out what type of target component during transformation by using transformation rule express.

Definition 3.2.2. (Physical Components) A Physical components is 3-tuples,

$$\mathcal{PC} =< N_{ode}, PP, PL >$$

where, N_{ode} is a execution platform, named *node* in Arcadia, it could be different type of physical component (e.g, processor, board). PP is the physical component port. PL is physical link, it could be assigned a concrete type such as *bus*.

Figure 4 is shown as a part of physical instance model of vehicle traction control unit in ARCADIA. We can see the yellow parts are the physical node (N_{ode}) and the red line is the physical link (PL) named bus in this case which connects to two physical ports (PP), the small square in dark yellow.

3.2.3 Physical elements transformation rules

According to the table 2, we can easily write the transformation rules for physical elements. Listing 3 shown as a part of the code to transform the physical component from Arcadia to AADL.

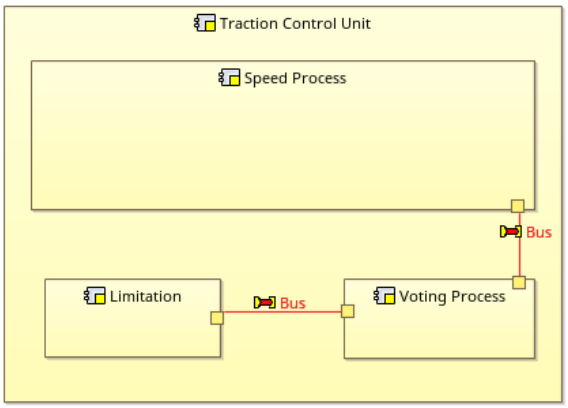


Figure 4: An example of physical view of vehicle traction control unit in ARCADIA

```
    columns
1 ΓN_ode ⇝ [Device|Process|Memory|Bus]:{
      Dispatch_Protocol}+:{Period}:{
      Deadline}+:{priority}+;
2 ΓPP ⇝ ¬PP;
3 ΓPL ⇝ Bus/BusAccess:[{
      Allowed_Connnection_Type}+:{
      Allowed_Message_Size}+|{
      Allowed_Physical_Access}+:{
      Transmission_Time}+];
```

Listing 3: Physcial elements transformation rules example

What we have to especially explain is the physical link part (see line 3). The Bus device could be a logical resource or hardware component. Hence, the bus device has different properties depending on the role. When the bus is considered as a logical resource, it contains the properties *Allowed_connection_type and Allowed_Message_Size*. When the bus is hardware, it contains *Allowed_Physical_Access and Transmission_Time*. Therefore, we write the rules that either

$\{Allowed\_Connnection\_Type\}+:\{Allowed\_Message\_Size\}+$

or

$\{Allowed\_Physical\_Access\}+:\{Transmission\_Time\}+$

4 Case Study

To show the efficacy of our approach in transforming and using produced AADL models to analyze the properties, this section presents the experimental results of analyzing the traction controlling unit of railway signaling system. By using our proposed approach, we transfer and extend Arcadia metamodel, and design AADL using OSATE2 with the generated metamodel. once the concrete models have been created, the scheduling property is chosen to show analysis ability through Cheddar tool (Singhoff et al., 2004).

4.1 Train Traction Control System

Train movement is the calculation of the speed and distance profiles when a train is traveling from one point to another according to the limitations imposed by the signaling system and traction equipment characteristics. As the train has to follow the track, the movement is also under the constraints of track geometry, and speed restrictions and the calculation becomes position-dependent. The subsystem of calculating the traction effective and speed restrictions is therefore critical to achieving train safe running. Nowadays, Communication Based Train Control (CBTC) system is the main method of rail transit (both urban and high-speed train) which adopts wireless local area networks as the bidirectional train-ground communication (Zhu et al., 2009). To increase the capacity of rail transit lines, many information-based and digital components have been applied for networking, automation and system inter-connection, including general communication technologies, sensor networks, and safety-critical embedded control system. A large number of subsystems consisting of modern signaling systems of railways, therefore, system integration is one of the key technologies of signaling systems; it plays a significant role in maintaining the safety of the signaling system (Wang and Wang,).

This paper uses a subsystem which called Traction Control Unit system (TCU) from signaling system of high-speed railway. We use this TCU system to illustrate the model transformation from engineering level to detailed architectural level and verified the instance models. The functional modules such as calculation and synchronization will be transformed using our approach, and then non-functional properties such as timing correctness and resource correctness will be verified by schedule analysis tool Cheddar (Singhoff et al., 2004).

First, we start with component functional views and physical view analysis by designing system models in Arcadia (shown in figure of TCU 5). The functions of the traction control system are to collect the external data by sensors such as a speed sensor. The data from Balise sensors is used to determinate the track block, and then it is going to seek the speed restriction conditions by matching accurate positioning (if the track blocks are divided fine enough) and digital geometric maps data. Meanwhile, calculating speed unit received the speed data from GPS and speed control commands from HMI (Human-Machine Interface) periodically. GPS data provides speed value periodically (we set a period of 30 seconds in this case), and HMI data sustainedly send the operation command with the period of 20 seconds till the value changed (e.g., expected speed value), then the calculating unit has to output an acceleration value and export to the locomotive mechanical system. Although they are periodic, the external data do not always arrive on time due to transmission delay or jitter. Therefore, we should use a synchronizer to make sure they are synchronized. Otherwise, the result would be wrong with asynchronous data. Similarly, to ensure the correctness of the command of acceleration (or deceleration), we applied a voting mechanism which can ensure the result is correct as much as possible. The voter must have the synchronized signal and restriction condition to dedicate to output the acceleration coefficient request to the locomotive system. The AADL diagram is shown in figure 6.

4.2 Model transformation

Using the Arcadia2AADL tool, the metamodel of the TCU system in Capella is translated into the corresponding AADL metamodel with the rules and approach which describes in section 3. For instance, on the one hand, the function class is translated into the thread in AADL. To analyze the timing properties, several attributes also have been added such as protocol type, deadline, execution time, period.

On the other hand, the physical part element Node translates to the processor in this case. Differ from simple physical Node in Arcadia; the processor element attaches rich properties such as scheduling protocol (scheduler type), process execution time. The allocation relationships on both physical and functional parts are translated into AADL as well.

4.3 Schedule verification

The external data and internal process work sequentially is an essential safety requirement of the system, and each task should be scheduled properly. How-

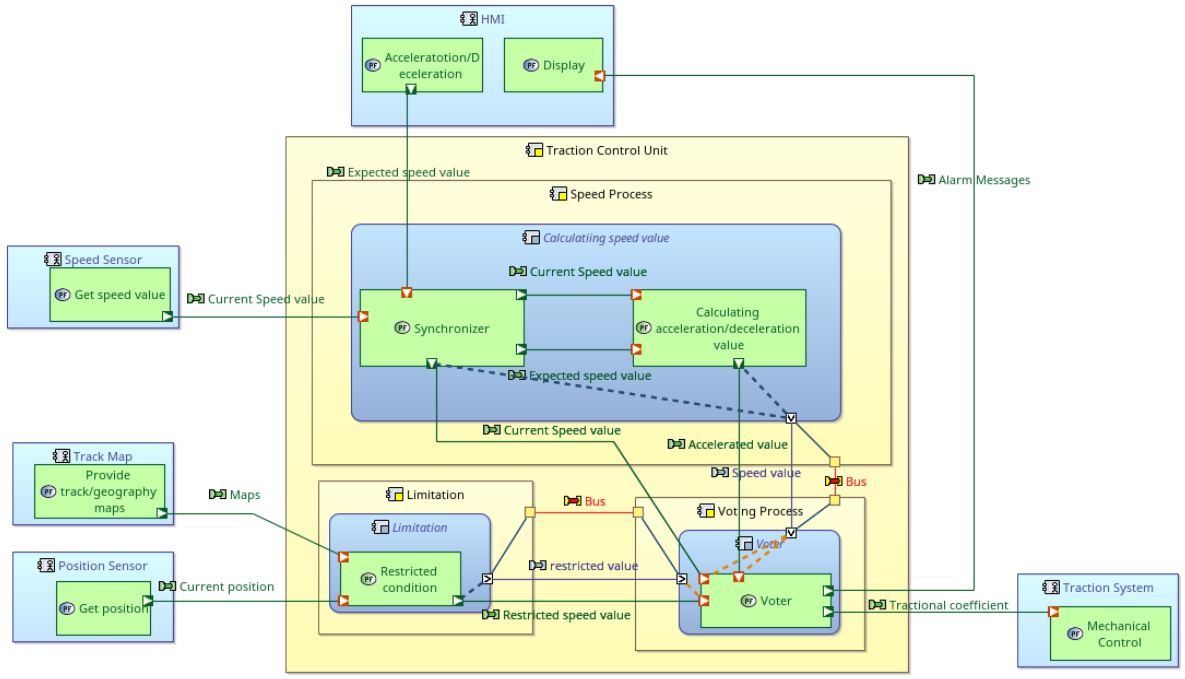


Figure 5: Arcadia model of TCU system

ever, in real-world, the risk of communication quality and rationality of scheduling must be taken into account. Therefore, the schedule verification is a way to evaluate system timing property. An Ada framework called Cheddar which provides tools to check if a real-time application meets its temporal constraints. The framework is based on the real-time scheduling theory and is mostly written for educational purposes (Marcé et al., 2005).

columns

```
1  thread implementation synchronizer.impl
2          properties
3          Dispatch_Protocol => perodic;
4          Period => 100 ms;
5          Deadline => 100 ms;
6          Compute_Execution_Time =>
               50..60ms;
7  end synchronizer.impl;
8
9  thread implementation calalculating.
       impl
10         properties
11         Dispatch_Protocol => perodic;
12         Period => 100 ms;
13         Deadline => 100 ms;
14         Compute_Execution_Time => 30ms
               ..40ms;
15 end calalculating.impl;
16
17 thread implementation gps.position
18         properties
19         Dispatch_Protocol => perodic;
20         Period => 40 ms;
21         Deadline => 40 ms;
22         Compute_Execution_Time => 30ms
               ..40ms;
23 end gps.position;
24
25 thread implementation HMI.setting
26         properties
27         Dispatch_Protocol => perodic;
28         Period => 30 ms;
29         Deadline => 30 ms;
30         Compute_Execution_Time => 20ms
               ..30ms;
31 end HMI.setting;
```

Listing 4: Setting of scheduling properties

Listing 4 shows a set of 4 periodic tasks (cal, pos, sync and setting) of TCU respectively, defined by the periods 100, 100, 40 and 30, the capacities 60, 40, 30 and 20, and the deadlines 100, 100, 40 and 30. These tasks are scheduled with a preemptive Rate Monotonic scheduler (the task with the lowest period is the task with the highest priority).

For a given task set, if a scheduling simulation displayed XML results in the Cheddar. One can find the concurrency cases or idle periods (see left of figure 7, comprise the software part and physical device part). People change the parameters directly and reload simulation; a feasible solution can be applied instead. After tuning, finally, the appropriate setting has displayed as in right of figure 7. According to this sim-

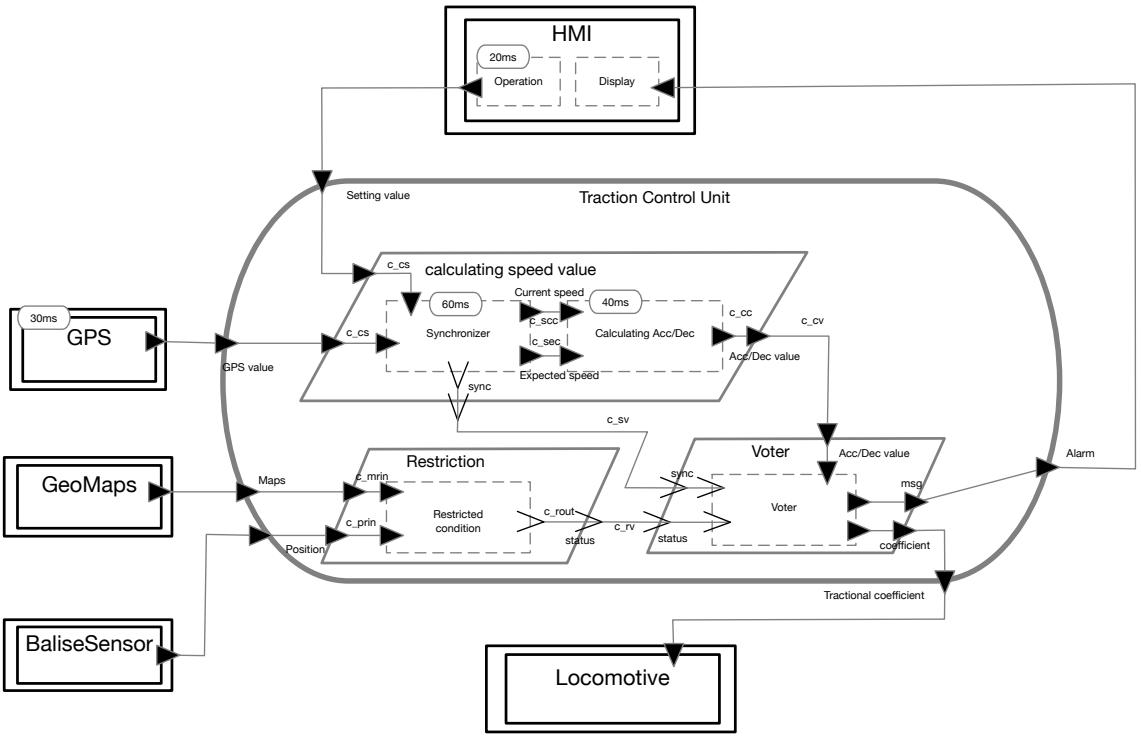


Figure 6: AADL model of TCU system

ulation result, people can correct the properties value in AADL, thereby ensure the correctness of system behavior timing properties.

5 Related work

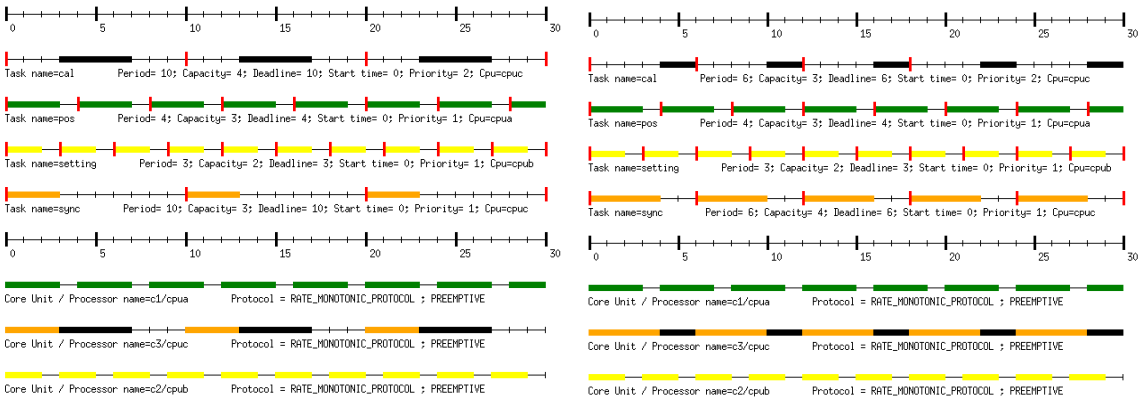
We have presented our approach to extending SysML-based engineering framework Capella to AADL and analyzed the relationships among Arcadia and AADL models in different view at the metamodel level. Likewise, a considerable number of studies have been proposed on ”language extension, modeling languages integration and composable language components”. This section provides a brief introduction to these works.

The complexity of the development of CPS has been the significant problems which puzzle the developers. It is not only from the nature of problems but also from the develop languages. Elaasar et al. has discussed (Elaasar et al., 2018) about the limit of UML which exacerbate the complexity of development, and proposed an approach to reduce the complexity of UML tools by implementing and adapting the ISO 42010 standard on architecture description.

Efficient integration of different heterogeneous modeling languages is essential. Modeling language integration is onerous and requires in-depth conceptual and technical knowledge and effort. Traditional modeling language integration approaches require language engineers to compose monolithic language aggregates for a specific task or project. Adapting these aggregates to different contexts requires vast effort and makes these hardly reusable. Arne Haber et al (Haber et al., 2015) presented a method for the engineering of grammar-based language components that can be independently developed, are syntactically composable, and ultimately reusable.

In despite of existing a lot of studies on the combining SysML and AADL (De Saqui-Sannes and Hugues, 2012) or on the extending SysML with AADL (Behjati et al., 2011). Differ from the above studies, our approach dedicates to smoothly combine engineering platform Capella/Arcadia, AADL and its annex, and our approach can be easily applied to other languages through fine-tuning. In practice, one could design global system at a high level and then seamlessly refine the models within AADL and its annex for further analysis such as scheduling. In other words, our approach can properly extend Arcadia’s design and analysis capabilities to AADL, while essentially keeping its independence.

An approach for translating UML/MARTE detailed design into AADL design has proposed by Brun et al. (Brun et al., 2008). Their work focuses on the transformation of the thread execution and commu-



(a) Schedule 1 with idel time

(b) Schedule 2 with compact time

Figure 7: Simulation results of tasks schedule

nication semantics and does not cover the transformation of the embedded system component, such as device parts. Similarly, in (Turki et al., 2010), Turki et al. proposed a methodology for mapping MARTE model elements to AADL component. They focus on the issues related to modeling architecture, and the syntactic differences between AADL and MARTE are well handled by the transformation rules provided by ATL tool, yet they did not consider issues related to the mapping of MARTE properties to AADL property. In (Ouni et al., 2016), Ouni et al. presented an approach for transformation of Capella to AADL models target to cover the various levels of abstraction, they take into account the system behavior and the hardware/software mapping. However, the formal definition and rigorous syntactic of transformation rules are missed.

Behjati et al. describe how they combined SysML and AADL in (Behjati et al., 2011) and provided a standard modeling language (in the form of the ExSAM profile) for specifying embedded systems at different abstraction levels. De Saqui-Sannes et al. (De Saqui-Sannes and Hugues, 2012) presented an MBE with TTool and AADL at the software level and demonstrated with the flight management system. Both of their works do not provide the description in a formal way.

In industrial domain applications, Suri et al (Suri et al., 2017) proposed a model-based approach for complex systems development by separating the behavior model and execution logic of the system. Moreover, they used UML based languages to model system behavior and connected the behavior models to external physical API of CPS. It focuses on providing a solution for the modularity and interoperability issues related to Industry 4.0 from a systems integration viewpoint.

S. Apel et al (Apel et al., 2016) also studied on different model driven methods for heterogenic systems for Electric vehicle. They have tried to evaluate how model-driven engineering (MDE) combined with generative frameworks can support the transfer from platform independent models to deployable solutions within the logistical domain.

The work of Kurtev (Kurtev et al., 2017) is used in the x-ray machine, it provided a family of domain-specific languages that integrate existing techniques from formal behavioral and time modeling. F. Scippacercola (Scippacercola et al., 2015) have explored the application of model-driven engineering on the interlocking system (a subsystem of signaling system of the railway). They discussed how to reduce efforts and costs for development, verification, and validation in a critical system.

The modeling language scientists have proposed some specific methods to weave the models as well as metamodels formally such as (Jezequel, 2008), Degueule has proposed Melange, a language dedicated to merging languages (Degueule et al., 2015), and similar works like (Ramos et al., 2007). However, the structural properties are not supported.

Compared with current studies, the approach proposed in this paper has the following features:

1. A proper subset of AADL has been chosen as the transformation target including functional software composition, execution platform. We use it to describe continuous behaviors of Cyber-Physical System.
2. All of the transformations is considered at metamodel level, and then a synthesized metamodel can be used to create concrete AADL models for further analysis.
3. Transformation rules are formally defined, and

then it is readable by human and easier to verify the correctness of transformation.

6 Conclusions and future work

This paper describes a collaborative design approach between system engineering methodology Arcadia (based on SysML) and architectural design language AADL using transformation at metamodel level. We first present our approach and implementation procedures using ATL. Then, we give a formal description of the key modeling elements of Arcadia and AADL, respectively. Then translation rules from these Arcadia metamodels to AADL are formally defined. Finally, a case study of train traction controlling system is used to demonstrate the transformation from engineering concerned design into an architectural refinement design which can be further analyzed by scheduling properties. There are some drawbacks to use our approach, i) people has to spent times to learn the syntax of rules, and the writing of rule is error-prone. ii) the traceback function is not yet implemented automatically.

In our future work, we will try to build a graphic interface to write rules, and the writing errors of rule can be detected. We also have to implement the traceback of simulation results which is sketched in our workflow with the arrow in dotted line. The results must be used automatically by upstream modeling framework. To this end, we have to extract the critical information from cheddar outputting file and transform to an appended file of modeling tool which can be recognized by the tool and hint user in somehow. Secondly, we will study the translation rules for more elements of Arcadia and also for comprehensive SysML elements, even for others UML-like profiles such as MARTE. At the same time, we will continue to explore the AADL and its annex to support more analysis and formal verification of system design. Besides, the safety-critical systems have become a trend in industrial files. We will study the extension of AADL with verification of safety properties with transformation methodology.

REFERENCES

Apel, S., Mauch, M., and Schau, V. (2016). Model-driven engineering tool comparison for architectures within heterogenic systems for electric vehicle. In *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 671–676.

Behjati, R., Yue, T., Nejati, S., Briand, L., and Selic, B. (2011). Extending SysML with AADL concepts for comprehensive system architecture modeling. In *European Conference on Modelling Foundations and Applications*, pages 236–252. Springer.

Brun, M., Vergnaud, T., Faugere, M., and Delatour, J. (2008). From UML to AADL: an Explicit Execution Semantics Modelling with MARTE. In *ERTS 2008*.

De Saqui-Sannes, P. and Hugues, J. (2012). Combining SysML and AADL for the design, validation and implementation of critical systems. In *ERTS2 2012*.

Degueule, T., Combemale, B., Blouin, A., Barais, O., and Jezequel, J.-M. (2015). Melange: A meta-language for modular and reusable development of dsls. In *Conf on Software Language Engineering*, pages 25–36. ACM.

Elaasar, M., Noyrit, F., Badreddin, O., and Gérard, S. (2018). Reducing uml modeling tool complexity with architectural contexts and viewpoints. In *MODELSWARD*, pages 129–138.

Haber, A., Look, M., Perez, A. N., Nazari, P. M. S., Rumpe, B., Vlkel, S., and Wortmann, A. (2015). Integration of heterogeneous modeling languages via extensible and composable language components. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 19–31.

Jezequel, J.-M. (2008). Model driven design and aspect weaving. *Software and Systems Modeling*, 7(2):209–218.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., and Valduriez, P. (2006). a QVT-like transformation language. ACM, New York, USA.

Kurtev, I., Schuts, M., Hooman, J., and Swagerman, D.-J. (2017). Integrating interface modeling and analysis in an industrial setting. In *MODELSWARD*, pages 345–352.

Marcé, L., Singhoff, F., Legrand, J., and Nana, L. (2005). Scheduling and Memory Requirements Analysis with AADL. In *SIGAda*, pages 1–10. ACM.

Ouni, B., Gaufillet, P., Jenn, E., and Hugues, J. (2016). Model Driven Engineering with Capella and AADL.

Ramos, R., Barais, O., and Jezequel, J.-M. (2007). Matching model-snippets. In *Conf on Model Driven Engineering Languages and Systems*, pages 121–135. Springer.

Scippacercola, F., Pietrantuono, R., Russo, S., and Zentai, A. (2015). Model-driven engineering of a railway interlocking system. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 509–519.

Singhoff, F., Legrand, J., Nana, L., and Marcé, L. (2004). Cheddar - a flexible real time scheduling framework. *SIGAda*, pages 1–8.

Suri, K., Cuccuru, A., Cadavid, J., Gérard, S., Gaaloul, W., and Tata, S. (2017). Model-based development of modular complex systems for accomplishing system integration for industry 4.0. In *MODELSWARD*, pages 487–495.

Turki, S., Senn, E., and Blouin, D. (2010). Mapping the MARTE UML profile to AADL. In *ACES-MB*, pages 11–20.

Wang, J. and Wang, J. A New Early Warning Method of Train Tracking Interval Based on CTC. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–7.

Zhu, L., Zhang, Y., Ning, B., and Jiang, H. (2009). Train-ground communication in CBTC based on 802.11 b: Design and performance research. In *CMC'09*, pages 368–372. IEEE.