Collaborative design: Explore CPSs analysis capabilities of SysML-based tools with AADL and Hybrid Annex

***Abstract*—The design of Cyber-Physical Systems (CPSs) has become increasingly complex and challenging. The collaborative design is a potential approach to deal with the rise of the complexity of system design which attracts more and more attention. Collaborative design relied on Model-driven engineering (MDE) involving several modeling languages (such as SysML, AADL, etc.) and using principles of separation of concerns as well as domain-specific languages (DSL). It makes stakeholders from diverse domains to work in a coordinated manner on different aspects of the system. It could help in reducing the gap between heterogeneous domains and making diverse expertise work together to produce a coherent and complete system. Therefore, there are requirements for the collaborative design to allow different domain experts to work together. In this paper, we show how to use the coordinated metamodel approach in MDE as a systematic way to gather isolated domain models and cross-cutting concerns. In practice, we proposed a set of transformation operators to manipulating metamodels; it aims to enrich the capacities of the platform by blending different languages seamlessly, as well as perform verification and validation respectively at the high-level and early stage of development. We illustrate our approach through experiments in the train traction controlling system design process.**

I. INTRODUCTION

The design of Cyber-Physical Systems (CPSs) has become increasingly complex and challenging. The developer should face numerous aspects, and each aspect has their problem space and characteristics.

As the Model-Based Engineering (MBE) has emerged as a key set of technologies to engineer the complex system. Experts thus usually used models and metamodels as the common countermeasure to capturing and solving the problems. In recent years, a number of modeling languages target CPSs have been standardized, but, to the best of our knowledge, none of them provide the full range required to deal effectively with the kinds of CPSs's complex that we have encountered. Some of them, such as Systems Modeling Language (SysML) [1], a more general-purpose language, focus on the "big picture" requirements, functional and architectural views, whereas others, such as Architecture Analysis and Design Language (AADL) [2] address the more detailed platform-oriented and physical aspects of such systems.

Since Cyber-Physical Systems interact with surrounding physical environment frequently, and discrete, continuous and hybrid behaviors describe the physical part, fox example, the physical symptom used DEs or PDEs (Partial Differential Equations) to express. A more general case is the hybrid system which is the continuous dynamics of interactions between a control system and its environment, thus Ahmad et al [3] proposed a hybrid annex of AADL to intensify AADL's capability to describe hybrid systems. In this paper, we also considered HA (hybrid annex) as a part of the target model.

Although AADL was a sophisticated analysis language and existing IDE tools such as OSATE [4] which can describe and verify both functional and non-functional properties of AADL models, it still insufficient to cater comprehensive system design and the need of abilities for engineering [5]. Therefore, AADL must be used with the upstream modeling language such as SysML and supporting engineering environment like Capella/Arcadia [6][7].

Capella platform and its method ARCADIA (**ARC**hitecture **A**nalysis and **D**esign **I**ntegrated **A**pproach) are to some extent a SysML-like solution to design the architecture of complex systems using models. However, unlike SysML, it appears to be a domain-specific language (DSL) which was preferred in order to ease appropriation by all stakeholders, usually not familiar with general-purpose, generic languages such as UML or SysML. Rather than having modeling "experts" owning the model on behalf of systems engineers, the ultimate goal of Arcadia is to have systems engineers pay more attention to global system design.

It is seldom the case that one development platform or single language can adapt to all aspects with assumption one-size-fits-all. Due to the engineers involved various modeling languages to different model domains respectively, and the engineers who from the different domain, each one has domain-specific tools. It does not only result in the proliferation of languages and its extensions for describing a variety of CPSs's aspects but also make the design complexity of CPSs increase. Meanwhile, the gaps between languages and platforms brought additional problems such as coherent and consistent problems, which is exposed at integration and simulation stages and further augment the complexity, make it skyrocketing.

To reduce the gaps and eliminate inconsistencies between SymML–like tool Capella/Arcadia and architecture design language AADL and its hybrid annex (HA), also to benefit from both of their powerful capabilities and strong points, we present a suite of evolving combining approach. In despite of existing a lot of studies on the combining SysML and AADL [8] or on the extending SysML with AADL [5]. Differ from above studies; our

method dedicates to smoothly combine engineering platform Capella/Arcadia, AADL and its annex, in this way, one could design global system at a high level and then seamlessly refine the models within AADL and its annex. That can also properly extend Arcadia's capability, and make it can perform analysis of discrete behavior variation and continuous environmental domain.

This paper presents an approach was to characterize the discrete and continuous properties of CPSs by models and abstract metamodels, respectively. Then merge the two metamodels to generate new metamodel at high abstract level. The engineer can use this generated metamodel to create kaleidoscopic instance models which can be used to further analysis (e.g., timing, safety). In this way, it allows the original engineering platform is capable of scaling to a wider range of capabilities through the use of relevant languages. Compared with similar studies, our approach makes ***three following major contributions***:

- A proper subset of AADL and its annex are chosen as the target metamodel including the thread (scheduling, dispatching and execution), port connections, and hybrid behavior. ii) Partial components of Arcadia (conform to SysML) are abstracted as the metamodel. Both above metamodels are defined formally.
- A set of relationships (e.g., transforming, creating, ignoring), semantic definitions and corresponding rules are provided to help the integration engineer custom-tailor metamodel they needed.
- we implement the transformation process among metamodels and practice our approach with a toolchain.

The rest of this paper is organized as follows: in the next section, we present the state-of-the-art of engineering models transformation briefly. Then the summary of our approach process is presented, as well as the metamodels, and its formal definitions are given in the section III. Section IV gives a set of relationships, semantics and corresponding rules. Section V describes the implementation and using a case study of train traction controlling systems to demonstrate architecture and secluding analysis. Conclusion and future work presented in the last section.

II. Related work

A considerable number of studies have been proposed regarding extending UML-like profile to AADL and model transformation methods. This section provides a brief introduction to these works.

An approach for translating UML/MARTE detailed design into AADL design has proposed by Brun et al. [9]. Their work focuses on the transformation of the thread execution and communication semantics and does not cover the transformation of the embedded system component, such as device parts. Similarly, in [10], 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 [11], 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.

The scientists have proposed some specific methods to weave the models as well as metamodels formally such as [12], Degueule has proposed Melange, a language dedicated to merging languages [13], and similar works like [14]. However, the structural properties are not supported.

Behjati et al. describe how they combined SysML and AADL in [5] and provided a common modeling language (in the form of the ExSAM profile) for specifying embedded systems at different abstraction levels. De Saqui-Sannes et al. [8] presented an MBE with TTool and AADL at the software level and demonstrated with flight management system. Both of their works do not provide the description in a formal way.

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

1) Arcadia is chosen as the transformation source. Arcadia provides a broad view of system engineering as well as refined functional and physical views.
2) A proper subset of AADL and its hybrid annex have been chosen as the transformation target including functional software composition, execution platform and the Hybrid annex which is usually used to describe continuous behaviors in a Cyber-Physical System.
3) All of the transformations is considered at metamodel level, and then a generated synthesized metamodel can be used to create concrete AADL models for further analysis.
4) Translational rules were defined in good form formally, and then it is readable by human and easier to verify the correctness of transformation.

III. Approach to Arcadia and AADL

The workflow of our approach is shown as figure 1. Since on the one hand the Arcadia methodology and Capella modeling platform focus on high level of engineering, and on the other hand, the AADL focuses on structural modeling to describe concrete execution behaviors of components, its annex can be used to describe hybrid behaviors. Hence, in this paper, we focus on combining Arcadia and AADL model semantically using metamodels combination. To extend the semantics of Arcadia (with its profiles) and SysML to AADL, we proposed a set of operators to specify various transformation case and constitutes the

transformation rules (called Transformation Rule LIB). For Transformation from Arcadia to AADL formally, we proposed a couple of metamodels of the subset of AADL and ARCADIA respectively, and their formal definitions have been given as well. As Hybrid Annex is an extension used to describe discrete and continuous behaviors, its metamodel and formal definition are also provided. For the more detailed syntax and semantics of hybrid annex, the reader is referred to [3].

Based on our defined AADL and Arcadia metamodels, system designs in Arcadia variation information can be extracted and transformed into corresponding AADL metamodels, once the new metamodels have been generated, designers can create concrete models for conducting further analysis, in this paper, we use generated AADL models to simulate the scheduling of system. The verification of the AADL model and result of the simulation are both can trace back to original Arcadia models. It can directly help designer tuning the system models, correct the vulnerabilities and flaws and improves the system performance.

A. AADL and Hybrid Annex (HA)

1) Background of AADL: Architecture Analysis and Design Language (AADL) is used to describe the system model, including both software and hardware (execution platform) parts. An AADL model can model the system as a hierarchy of software components bound to an execution platform and conduct a system analysis. Predefined abundant components such as thread, thread group, process, data and subprogram in soft categories, and processor, memory, device, and bus in platform categories.

There are three kind of components in AADL:

1) Application software components: **data**, **subprogram**, **thread**, **thread group**, **process**,
2) Execution platform components: **processor**, **memory**, **virtual memory**, **bus**, **virtual bus**, **device**
3) Composite component: **system**

In AADL, each element in category of the component includes the *component type and component implementation*. The type is a specification of the external interface. A component type contains features, flow specifications, property associations. A component type defines the communicational port, while component implementation specifies subcomponent and the interactive among the subcomponents. The features of AADL are designed to model the communication between different components. It is composed of **event** port, **data** port, **event data** port, parameter access, data access and bus access. Each component contains some non-functional properties.

The **thread** component includes priority, preemptive, dispatch protocol, and time-related property. And **process** component is a composition unit of **thread** component, and includes all the resource of **thread** component. The component implementation must conform to their type which includes the detail contents such as refined type, sub-components, connections, flow, functional and non-functional properties.

2) Software Functional Composition: In AADL, application software components communicate each other and they rely on the processor, memory and bus of execution platform component. Even they may interactive with device components.

Definition 2.1. (Software Functional Composition) A **SFC** is a seven tuples:
$\mathcal{A} =< Type, Impl, \Sigma_{T,I}, Port, Property, Flow, Annex >$
where:

1) $Type$ defines the category of the components.
2) $Impl =< Sub, Conn >$ is the corresponding specific implementation of a component type. And each implementation includes the deployment of subcomponents and connections.
3) $\Sigma_{T,I} \subset Type \times Impl$ defines the relationship of component type and component implementation.
4) $Port$ is a set of communication unit among threads. Port includes **data** port, **event** port and **data event** port. And, port is divided into **in** port, **out** port, **in out** port.
5) $Property$ is a set of non-functional feature.
6) $Flow$ specifies **flow source**, **flow sink** and **end to end flow**.
7) $Annex$ is a set of the extensions of AADL.

3) Execution Platform: AADL component model is an abstract of actual systems. So, all the application software components finally run on execution platform component and the connections between application software component are bound to **bus** component.

Definition 2.2. (Execution Platform) A **EP** component is defined as a six tuples:
$\mathcal{E} =< Type, Impl, \Sigma_{T,I}, BusAccess, Property, Annex >$
where:

1) $BusAccess$ defines the interactive approach between **bus** component and other execution platform components.
2) $Property$ is a set of non-functional feature.
3) $Type$ defines the category of the components.
4) $Impl$ is the corresponding specific implementation of a thread type.
5) $\Sigma_{T,I} \subset Type \times Impl$ defines the relationship of component type and component implementation.
6) $Annex$ is a set of the extensions of AADL.

4) Binding: In AADL, the pre-defined property set includes *deployment_properties*, which is used to describe the deployment relationship from the software component to execution platform component. Here, we define *bind* as an operator between application software components and execution platform components.

Definition 2.3. (Binding) In the system with multiple processors, *bind* is a tuple: $\mathcal{B} =< SFC, EP, \Sigma B_{S,E} >$, where

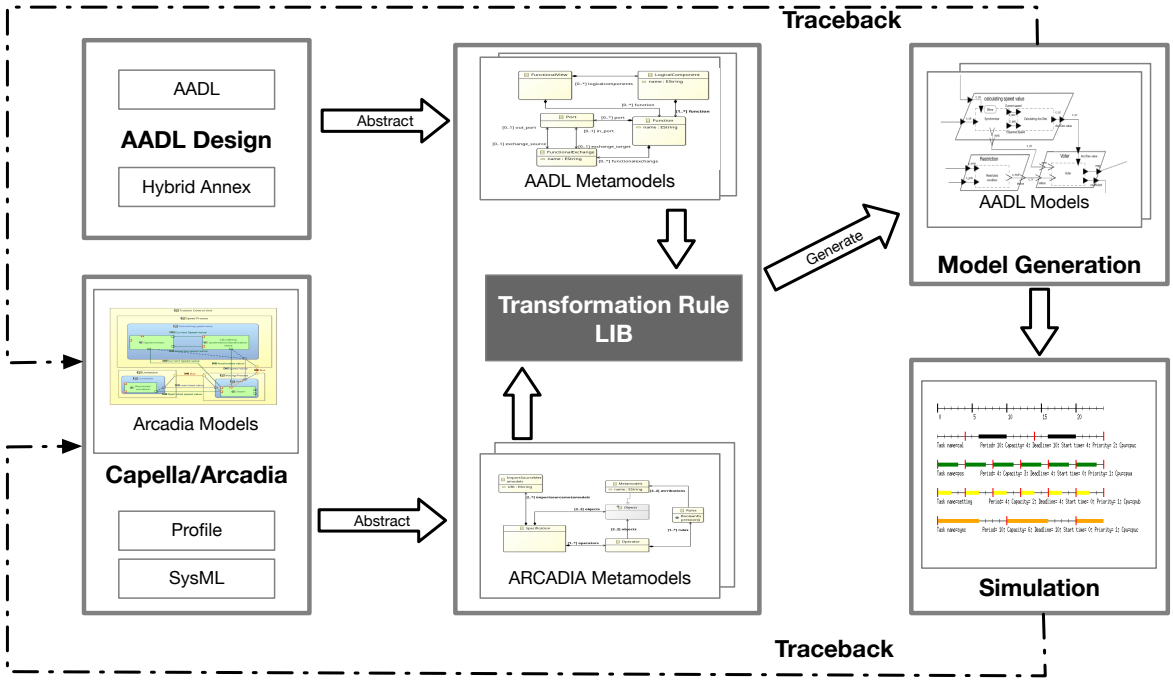


Fig. 1. Workflow

1) SFC is a set of application software components.
2) EP is a set of hardware components.
3) $\Sigma B_{S,E}$ is holistic binding relation space between software components and hardware components.

Multiple application software components can execute on the same **processor** component or **memory**, and multiple connections between application software components can be bound to the same **bus** component. However, at the same time, the execution platform has limited resources, it can only support software components with restricted rules. Over many request resources lead to conflicts.

5) Hybrid Annex: Hybrid Annex (HA) is a lightweight extension in order to model those physical, real-world elements, or processes. HA subclauses annotate either AADL device component implementations to model the continuous behavior of sensors and actuators, or abstract component implementations to model the continuous behavior of physical processes. Ehsan Ahmad et al. proposed an HA specification consist of six sections: *Assert, Invariant, Variables, Constants, Channels, and Behavior* [3]. *Assert* is used to declaring predicates which may be used with invariants to define a condition of operation. 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. 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 channels 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 interrupts, 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 2.4 gives a metamodel of Hybrid Annex which does not exist in SysML,

Definition 2.4. (Hybrid Annex) A *Hybrid Annex* is a 8-tuple $< Ass, Ivar, Var_{hd}, Cons_{hd}, P_{roc}, ChP, Itr, B_{itr} >$, where

1) Ass is a finite set of assert for declaring predicates applicable to the intended continuous behavior of the annotated AADL component.
2) $Ivar$ is associated with assert to define a condition of operation that must be true during the lifetime.
3) Var_{hd} is a finite set of discrete and continuous variables.
4) $Cons_{hd}$ is a finite set of constants which must be initiated at declaration.
5) P_{roc} is a finite set of processes that are used to specify continuous behaviors of AADL components.
6) ChP is a finite set of channels and ports for synchronizing processes.

7) Itr is a finite set of time or communication interrupts.
8) $B_{itr} : Itr \rightarrow P_{roc}$ binds interrupts to related processes.

B. Arcadia and SysML

In practice, the operational added value of the MBSE approach is based on many other criteria such as the definition of project modeling objectives, the implementation of adapted methods, the skills of the teams, the involvement of the hierarchy, the integration with the existing information system and third-party tools. In short, there are other aspects to consider when evaluating when to use a SysML tool or Capella than just the language. Arcadia presents a methodology to define, design, analyze and validate systems with software and hardware architecture. It provides a hierarchical structure with logical/functional components, physical components. Logical components deploy into physical components. Here, we define *allocate* as an operator to describe the relationship of functional components with physical components. Formally, an *allocate* operator is defined as a tuple: $< C_{logi}, C_{Phy} >$

1) Logical components: Blocks are modular units of system descriptions in SysML and are generalizations of the UML class concept. In the Arcadia, Functional chains diagrams consist of a set of SysML blocks and its interactions, named *Logical components*; The notion of Logical components in Arcadia 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 2.5, we present a metamodel of an instance of logical components.

Definition 2.5. (Logical Component) A logical component chain is 8 tuples, $< C_{omp}, FPC, P_{alloc}, Ex_{comp}, F_{un}, P_{ort}, Ex_{fun}, M_{cf} >$ where,

1) $C_{omp} = \{\bigcup_{F_{un}}\}$ is a logical component container which contains a set of functional blocks.
2) $FPC = \{\bigcup_{P_{port}}\}$ is a functional port container which contains a set of functional ports.
3) $P_{alloc} = \{FPC, PP\} : \Sigma FPC \rightarrow PP$ donates a finite set of allocated port which represents the allocable relationship between functional port container(FPC) and physical port (PP). In Arcadia, the source functional port container is named *Allocated function ports*, and the target physical port is named *Allocating physical ports*.
4) $Ex_{comp} \subseteq P_{alloc} \times P_{alloc}$ is a finite set of exchange between logical components.
5) F_{un} is a finite set of functional block include their name and id attributes.
6) P_{ort} is a finite set of functional ports including directions and allocation attributes.
7) $Ex_{fun} \subseteq P_{ort} \times P_{ort}$ donates a finite set of functional exchange (connection) between two functional ports, it must be pair, one is source, another is target.
8) $M_{cf} : \Sigma F_{un} \rightarrow C_{omp}$ allocate functions to a logical component container.

2) Physical components: The blocks can also represent hardware component in SysML, but in the Arcadia, there are the specific notation for physical components and its port, interaction so-called Physical link.

Definition 2.6. (Physical Component) A logical component chain is 3 tuples, $< N_{ode}, PP, Ex_{fpp}, PL >$ where,

1) N_{ode} is a execution platform, named *node* in Arcadia, it could be different type of physical component (e.g, processor, board).
2) PP is the physical component port.
3) $Ex_{fpp} \subseteq P_{alloc} \times PP$ is a finite set of exchange between functiotnal allocation port and physical ports.
4) PL is physical link, it could be assigned a concrete type such as *bus*.

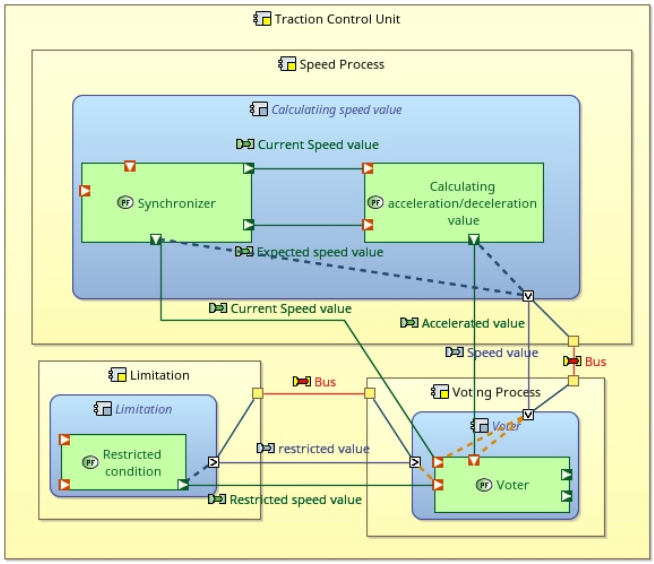


Fig. 2. An example of instantiate model of kernel unit of train traction control system in ARCADIA

In the figure 2, there is an instance model of kernel unit of train traction control system in Arcadia as an example. We can see the yellow parts are the physical node (N_{ode}) and the red line is the physical link (PL) which connects to two physical ports (PP) named bus in this case. A rectangle in blue is the logical component (C_{omp}) which contains the functional components (the rectangle in green F_{un}). 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}). A white square with a black arrow at the edge of the logical component is the allocated port, and the dotted line in green is the allocatable relation (P_{alloc}) which represents the allocation between functional port containers and physical ports, and the black line is the exchange link (Ex_{comp}) between two port allocations.

IV. Transformation

Each component in Arcadia that we mentioned in the above section is to be translated into counterpart in AADL (e.g., functional block, port, connection, physical node). Since our approach focuses on generating AADL models for further analysis, during the transformation procedure, we theoretically neglect all the Arcadia attributes or parts of the components that exist only in Arcadia but could not find the corresponding object in AADL. On the other hand, some of the missing components and attributes in Arcadia are complemented when translated to AADL. This section presents the translation rules and the operation. In the following parts of the paper, several symbols are defined for the convenience of formal description of transformation rules (see table I), in which: one rule beginning from Γ and end with ";"; the parent node is enclosed within the angle brackets "< >" (if it has one parent node); the attribute group are enclosed with parentheses "{ }"; square braces "[ ]" delimit optional elements and the alternatives are separated by a pipe "|"; each part of object separated by ":"; and the parentheses with plus "{ }+" is used to present the option is to be created; for some ignored attributes and objects are donated "¬" which is in front of ignored object. An example of transformation rule expressions is in listing 2. What we have to note is each rule can contain one source object which in the left of transfer symbol ⇝ and one or more attributes, and in the right side is target object and its member attributes. 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.

Symbol	Meaning
Γ	Transformation Rule
;	End of rule
⇝	Transfer
<>	Parent node
{ }	Attribute
[ ]	Optional element
|	Separation of elements
{ }+	Attribute is to be created
¬	Ignore

TABLE I
Symbols of transformation rule expression

A. Port and Connection

Ports are the logical connection points between components. AADL defines three types of component ports, for the transmission of data by data port, control information by event port and both of them by data event port. There are two directions of port, input, and output. The output port is connected to the input port to constitute the port connection. On the other hand, Arcadia defines only directions (in and out), the type of port is omitted. Hence, we ought to add the type attribute to complete the form in AADL when doing a transformer. The translating rule writes as an example in list 2 at line 1. It means the transformation of one functional port P_{ort} of Arcadia to a port of AADL (within the parent node *<feature>*). The direction attribute and its values *in or out* can transfer to counterpart directly, and the data type is additional option, it will be add with its values *data, event, data event*, denoted *{Type[data|event|data event]}+*. For some attribute which does not exist in AADL such as *ordering* (see list at line 3), we can write one line with the symbol ¬, it means the ordering attribute will be ignored for transformation.

A connection is an interaction between two objects via ports. One connection must have only one *source* and *target*. It is the same in both Arcadia and AADL. An example of transformation expression is shown in line 4.

However, all the transformation rule expression depends on the needs of engineering, if designers need some of the attributes to carry out analysis in next steps, they can add that they need, otherwise, they can write one line of expression to declare to ignore or omit it.

```
1 ΓPort:{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 ΓPort:¬{ordering});
4 ΓExfun:{Source}:{Target} ⇝ <connections>:
    connection:{source}:{target};
```

Listing 2. An example of transformation rules

B. Logical components

The logical components in Arcadia contain a set of member elements, such as logical component containers, functions, ports, and functional exchanges. In order to find the counterpart in AADL, we give an explicit correspondence table according to metamodels of Arcadia and AADL (see the formal definition of the logical component of Arcadia 2.5 and formal definition of software functional composition in AADL 2.1). In other words, the designer can combine the functional subset of Arcadia with the software functional composition subset of AADL using their metamodels.

```
system TractionControlUnit
       features
              speed: in data port speed;
              maps: in data port maps;
              position: in data port position;
              acc_coef: out data port votingResult;
              alarm: out data port votingResult;
              acceleration: in event data port command;
              ...
end TractionControlUnit;

system implementation TractionControlUnit.impl
       subcomponents
              computing: process computation.impl;
              Voting: process voting.impl;
              restrictedCondition: process RestrctedCondition.impl;
       connections
              c_sc: port speed -> computing.speed;
              c_ac: port acceleration -> computing.acceleration;
              c_mrin: port maps -> restrictedCondition.maps;
              c_prin: port position -> restrictedCondition.position;
              c_rv: port restrictedCondition.restriction -> Voting.restriction;
              c_cv: port computing.result -> Voting.result;
              c_vd: port Voting.outOfRange -> alarm;
              c_vl: port Voting.acc_coef -> locomotive;
              ...

annex hybrid {**
              assert
                     << SafeSpeed :: (speed@now < limit_speed) and (speed@now >= limit_speed) >>
              invariant
                     << SafeSpeed () >>
              variables
                     P : TS :: TractiveEffort --the tractive effort of the locomotive;
                     Q : TS :: Drag --the drag in newton;
                     B : TS :: Brake --the brake force;
                     a : TS :: Angle --the angle of train
                     alpha : TS :: Gradient --the gradient
                     limit_speed : TS :: limitSpeed --max speed of train
                     s : TS :: Speed --current speed
              constants
                     m= 2000 -- mass value
              behavoir
                     alpha ::= sin(a)
                     Acc ::= 'DT 1 m*dv/dt=(P-Q-m*g*alpha)' [[> out_coef ]]> Next
                     Des ::= 'DT 1 m*dv/dt=(P-(Q+B)-m*g*alpha)' [[> out_coef ]]> Next
                     Next ::= skip
                     Traction ::= speed_p?(s) & (s>limit_speed) -> Des
                     **};
end TractionControlUnit.impl;
```

Listing 1. Train traction control unit implementation with hybrid annex specifications in AADL

Arcadia	AADL	Additional attributes
logical component container (C_{omp})	process, system	{Runtime_Protection[true|false]}+
functional port container (FPC)	feature group	◯
port allocation (P_{alloc})	binding	◯
exchange of logical component (Ex_{comp})	binding	◯
function (F_{un})	thread, abstract	{Dispatch_Protocol[Periodic|Aperiodic|Sporadic|Background|Timed|Hybrid]}+
port (P_{ort})	port	{Type[data|event|data event]}+
exchange of function	connection	◯
allocation	binding	◯

TABLE II
FUNCTIONAL ELEMENTS CORRESPONDING TABLE

C. physical component

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. The table III given an example of the corresponding relation of the physical component between Arcadia and AADL.

D. Hybrid Annex

When transforming the behavioral description to a Hybrid annex, one must provide some necessary elements like variables, invariants, constants, channels, and behavior. Since Hybrid annex adopts the process algebra notations, the behavior of components is described by a set of CPS process; mostly it uses the differential equation for a continuous process. For example, the acceleration and deceleration in the traction control system of the train, refer to listing 1.

During the transformation, the continuous evolution of a CPS processes is expressed by some differential equations, which would be an attribute of the functional component in Arcadia. As an example shown in Listing 1, the differential expression 'DT 1 m*dv/(P-Q-m*g*α)' describes the relationship between distance and acceleration time, where m is mass of train, P is the tractive effort of the locomotive, Q is the drag in Newton, alpha is the gradient. the differential equation is written as

$$m\frac{dv}{dt} = P(v) - Q(v) - mg\alpha$$

which, once the initial speed is given, defines v as a function of time t. For the deceleration is the same, one just considered time the breaking factor. However, there are still some correlated variables are not declared in Arcadia, in this case, the designer has to add (create) them in Hybrid Annex code.

```
1 Γ[F_un|C_omp]:equations ⇝ <[abstract|system|
    process|thread]>:<hybrid annex>:behavior;
2 Γ[F_un|C_omp] ⇝ <[abstract|system|process|thread
    ]>:<hybrid annex>:{assert}+:{invariant}+:{
    variables}+:{constants}+;
```

Listing 3. Translating of Hybrid Annex

E. Implementation of Arcadia2AADL transformation tool

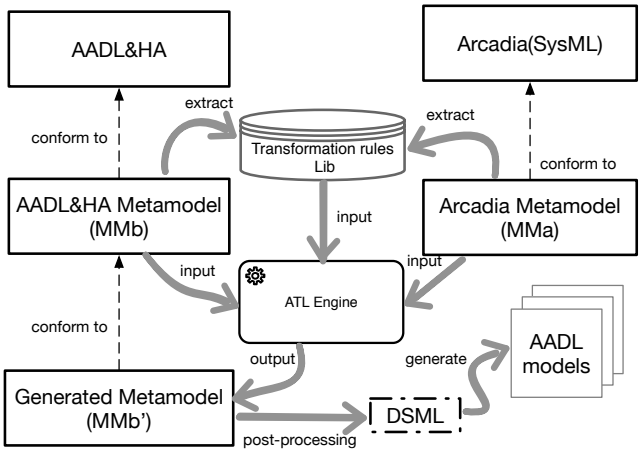


Fig. 3. Structure of Arcadia2AADL model transformation tools

Based on the translation rules defined in section IV, we implemented Arcadia2AADL transformation tool which can generate target metamodel according to original metamodels and transformation rule library. The ATL serves as a transformation engine. For more detailed information of ATL, the reader is referred to [15][16]. The tool architecture is shown in Fig 3. AADL&HA (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) the corresponding metamodel (denoted as generated metamodel MMb') which is a subset of AADL&HA, in other words, the generated metamodel is fit AADL and its hybrid annex. The generated metamodel is then further post-processed as a DSML (domain-specific modeling language) in *ecore* format. Which can be used to create concrete models. Therefore, all concrete models afterward are conformed to this metamodel and will be used for further verification and analysis.

V. CASE STUDY

To show the efficacy of our approach in transforming and using produced AADL models to analyze the prop-

Arcadia	AADL	Additional attributes
physical component (N_{ode})	device,memory,processor	{Dispatch_Protocol}+:{Period}:{Deadline}+:{priority}+
physical component port (PP)	port	◯
exchange of logical port and physical port (Ex_{fpp})	binding	◯
physical link (PL)	bus	◯

TABLE III
PHYSICAL ELEMENTS CORRESPONDING TABLE

erties, 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 [4] with the generated metamodel. once the concrete models have been created, the scheduling property is chosen to show analysis ability through Cheddar tool.

A. Train traction controlling system (TCU)

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) 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 [17]. To increase the capacity of rail transit lines, many information-based and digital components have been applied for networking, automation and system interconnection, including general communication technologies, sensor networks, and safety-critical embedded control system.

This paper uses a subsystem called traction controlling system (TCU) from signalling system of high-speed railways to illustrate the model transformation from engineering level to detailed architectural level and verification of instance models. 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 simulation tool Cheddar [18].

The functions of the traction control system are to collect the external data by sensors such as 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, and HMI data send the operation command (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 shown as figure 4.

B. 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 IV. 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.

C. 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. However, 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 [19].

```
1 thread implementation synchronizer.impl
2       properties
3       Dispatch_Protocol => perodic;
```

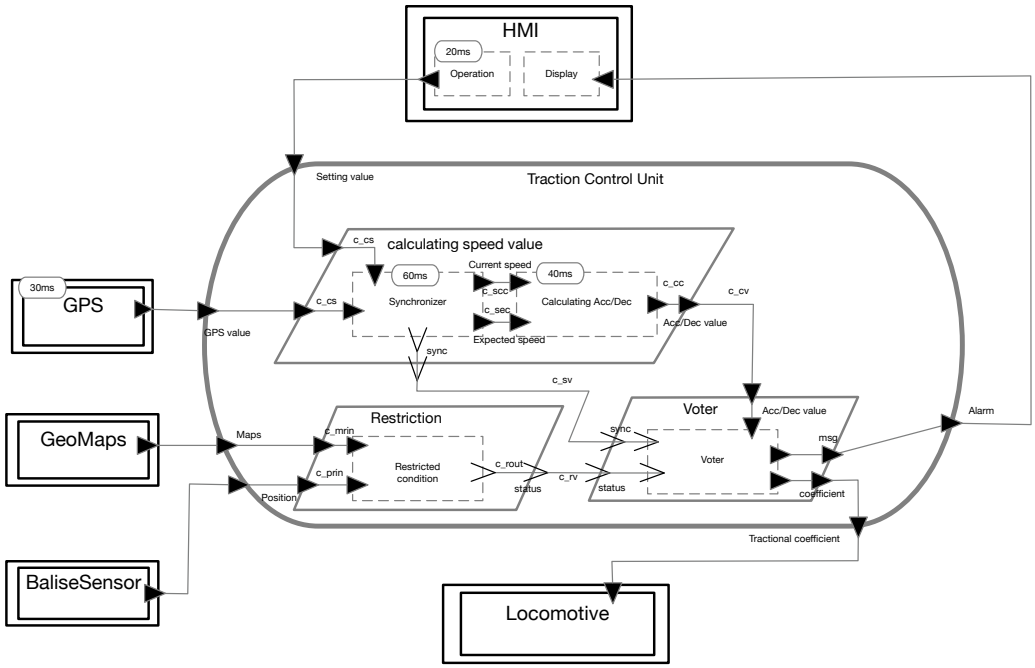


Fig. 4. AADL model for TCU of signaling system

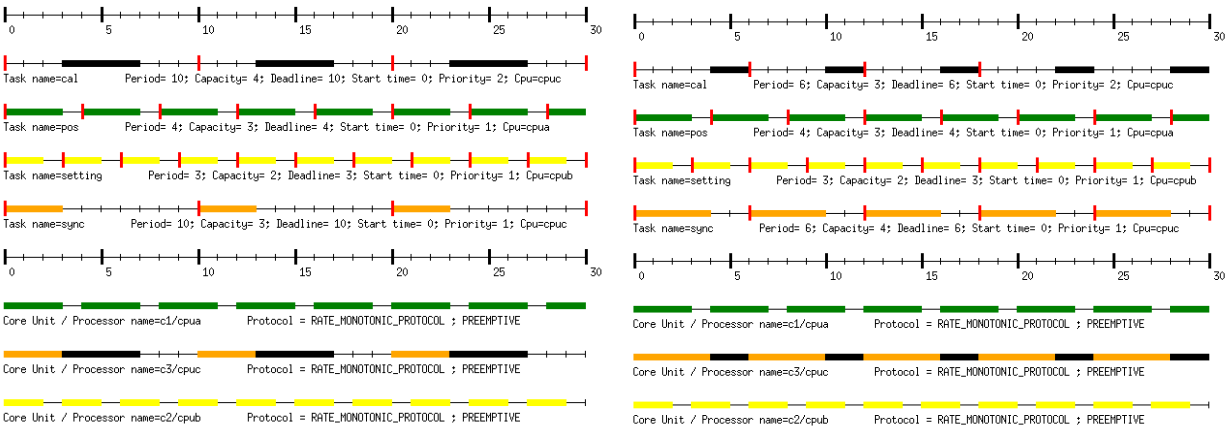
```
4       Period => 100 ms;
5       Deadline => 100 ms;
6       Compute_Execution_Time => 50..60ms;
7 end synchronizer.impl;
8
9 thread implementation calalculating.impl
10      properties
11      Dispatch_Protocol => perodic;
12      Period => 100 ms;
13      Deadline => 100 ms;
14      Compute_Execution_Time => 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. An example of setting of scheduling properites

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 5, comprise the software part and physical device part). People change the parameters directly and reload simulation; an feasible solution can be applied instead. After tuning, finally, the appropriate setting has displayed as in right of figure 5. According to this simulation result, people can correct the properties value in AADL, thereby ensure the correctness of system behavior timing properties.

VI. Conclusions and future work

This paper describes a collaborative design approach between system engineering methodology Arcadia (based on SysML) and architectural design language AADL (include hybrid annex) using transformation at metamodel level. We first present a formal description of key modeling elements of Arcadia, AADL, and hybrid annex, respectively. Then translation rules from these Arcadia metamodels to AADL are formally defined. Finally, we give the implementation procedures using ATL, and a case 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 on scheduling properties. In our future work, 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



(a) Schedule 1

(b) Schedule 2

Fig. 5. Simulation results of tasks schedule

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

[1] O. M. Group, "OMG Systems Modeling Language," May 2017.

[2] "Model-Based Engineering with AADL," 2012.

[3] E. Ahmad, B. R. Larson, S. C. Barrett, N. Zhan, and Y. Dong, "Hybrid Annex: An AADL Extension for Continuous Behavior and Cyber-physical Interaction Modeling," *Ada Lett.*, vol. 34, no. 3, pp. 29–38, Oct. 2014.

[4] OSATE2. [Online]. Available: http://osate.github.io/

[5] R. Behjati, T. Yue, S. Nejati, L. Briand, and B. Selic, "Extending SysML with AADL concepts for comprehensive system architecture modeling," in *European Conference on Modelling Foundations and Applications*. Springer, 2011, pp. 236–252.

[6] Capella. (2014) Capella. [Online]. Available: http://www.polarsys.org/capella

[7] *A Model-Based Engineering Method for System, Software and Hardware Architectural Design*, Nov. 2016.

[8] P. De Saqui-Sannes and J. Hugues, "Combining SysML and AADL for the design, validation and implementation of critical systems," in *ERTS2 2012*, 2012, p. 117.

[9] M. Brun, T. Vergnaud, M. Faugere, and J. Delatour, "From UML to AADL: an Explicit Execution Semantics Modelling with MARTE," in *ERTS 2008*, 2008.

[10] S. Turki, E. Senn, and D. Blouin, "Mapping the MARTE UML profile to AADL," in *Proceedings of the 3rd International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB 2010)*. Citeseer, 2010, pp. 11–20.

[11] B. Ouni, P. Gaufillet, E. Jenn, and J. Hugues, "Model Driven Engineering with Capella and AADL," p. 0, Jan. 2016.

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

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

[14] R. Ramos, O. Barais, and J.-M. Jezequel, "Matching model-snippets," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2007, pp. 121–135.

[15] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, *ATL: a QVT-like transformation language*, ser. a QVT-like transformation language. New York, USA: ACM, Oct. 2006.

[16] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31–39, Jun. 2008.

[17] L. Zhu, Y. Zhang, B. Ning, and H. Jiang, "Train-ground communication in CBTC based on 802.11 b: Design and performance research," in *Communications and Mobile Computing, 2009. CMC'09. WRI International Conference on*. IEEE, 2009, pp. 368–372.

[18] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar - a flexible real time scheduling framework." *SIGAda*, pp. 1–8, 2004.

[19] L. Marcé, F. Singhoff, J. Legrand, and L. Nana, "Scheduling and Memory Requirements Analysis with AADL," in *Proceedings of the 2005 Annual ACM SIGAda International Conference on Ada*. New York, NY, USA: ACM, 2005, pp. 1–10.