Front.Comput.Sci.
DOI : 10.1007/s11704 – 018 – 7039 – 7

RESEARCH ARTICLE

AADL+ : A Simulation – Based Methodology for Cyber – Physical Systems

Jing LIU[1], Tengfei LI[1], Zuohua DING[2], Yuqing QIAN[1], Haiying SUN[1], Jifeng HE (✉)[1]

1 Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China
2 Zhejiang Sci-tech University, Xiasha College Park, Hangzhou 310018, China

Abstract AADL (Architecture Analysis and Design Language) concentrates on the modeling and analysis of application system architectures. It is quite popular for its simple syntax, powerful functionality and extensibility and has been widely applied in embedded systems for its advantage. However, it isn't enough for AADL to model cyber-physical systems (CPS) mainly because it cannot be used to model the continuous dynamic behaviors. This paper proposes an approach to construct a new sublanguage of AADL called AADL+, to facilitate the modeling of not only the discrete and continuous behavior of CPS, but also interaction between cyber components and physical components. The syntax and semantics of the sublanguage are provided to describe the behaviors of the systems. What's more, we develop a plug-in to OSATE (Open-Source AADL Tool Environment) for the modeling of CPS. And the plug-in supports syntax checking and simulation of the system model through linking with modelica. Finally, the AADL+ annex is successfully applied to model a lunar rover control system.

Keywords AADL, cyber-physical systems (CPS), simulation, OSATE, lunar rover control system

1 Introduction

Cyber-physical systems are consolidations of computation, communication, and control processes, putting emphasis on the interaction between cyber components and physical components, and have become the next generation intelligent system which has the capability of autonomous perception, adjustment, and judgment. They also accomplish the interconnection between virtual world and physical world [1]. Modeling such systems with reasonable fidelity is a challenge since it needs technologies of control engineering, software engineering, and sensor networks. We need to consider not only the functionality of the system, but also the reasonable resource allocation of the system, the system performance optimization, personalized services and user experience. There are some modeling languages such as UML (Unified Modeling Language) [2], MARTE (Modeling and Analysis of Real-Time and Embeded Systems) [3], SysML (Systems Modeling Language) [4], AADL [5], etc. Among them, AADL is a popular architecture modeling language, which is originally from Avionics Architecture Description Language and bases on similar language MetaH [6]. AADL has a strong modeling capability and has been applied in the fields of avionics, aerospace, automotive, robotics communities, etc. In this paper we propose a modeling language for CPS based on AADL.

1.1 The background of CPS

A cyber-physical system is an integration of information world and physical process. It is applied extensively in embedded system, Internet of things (IoT), cloud computation, wireless sensor network (WSN) and Industry 4.0 and other currently popular fields [7]. A cyber-physical system has two interactive layers: cyber layer and physical layer. Physical layer contains some physical entities with mutual associations. These physical entities connect interactively. Changes

Received February 09, 2017; accepted November 02, 2017

E-mail: *jifeng@sei.ecnu.edu.cn*

in a physical entity may lead to corresponding variations in other entities. While cyber layer consists of multiple intelligent monitoring nodes (including servers, mobile equipments, person and so on) and the communication among them [8]. The interaction of information and strategy decision is implemented through the 3C (Computation, Communication and Control) under the action of physical layer and cyber layer.

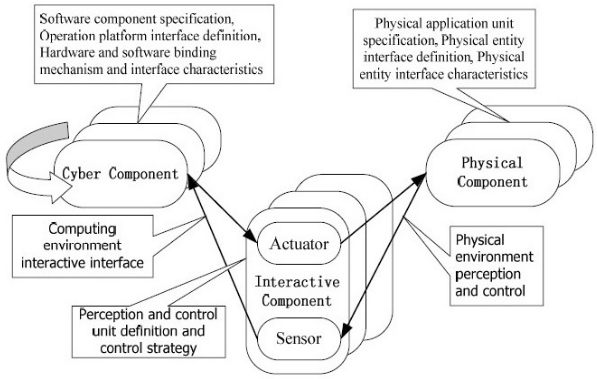

Software component specification, Operation platform interface definition, Hardware and software binding mechanism and interface characteristics
Physical application unit specification, Physical entity interface definition, Physical entity interface characteristics
Cyber Component
Physical Component
Computing environment interactive interface
Actuator
Interactive Component
Sensor
Physical environment perception and control
Perception and control unit definition and control strategy

Fig. 1 Component abstraction of CPS.

The component abstraction of cyber-physical systems should include three categories: cyber component, physical component and interactive component between them. Fig. 1 shows the component abstraction of CPS. The interactive component includes sensors and actuators. Sensors collect corresponding data from physical environment, while actuators respond to the physical environment according to computing results from cyber world. To model a cyber-physical system, it is indispensable to model the above three sorts of components.

1.2 AADL preliminary

AADL provides lots of modeling concepts with corresponding semantics. Its core language contains a lot of modeling concepts and meta models of many fields. More importantly, the core language is extendable, hence, in order to customize a sublanguage according to the corresponding requirements, users can extend the core language of AADL, i.e., an annex enables users to extend AADL and allows to include customized notations in a standard AADL model [9]. For instance, the language that is able to analyze a critical aspect of system (i.e. reachability analysis, reliability analysis and security) can be incorporated into standard AADL language.

There are two ways to extend the core language of AADL:

- Introducing new properties or notations.
- Using approved sublanguage extensions.

In the second way, this kind of sublanguage extensions of AADL is called annex. The following are the approved AADL annexes: Error Model Annex, Behavioral Annex [10], Graphical AADL Notation Annex, Programming Language Compliance and API Annex [11].

Right now AADL is not able to support modeling cyber-physical systems mainly because it isn't adaptive to describe continuous dynamics. For this reason, we propose an extension to AADL, called AADL+, to support the modeling and simulation of cyber-physical systems. The purpose of proposed AADL+ is to provide a comprehensive framework for the formal specification of cyber-physical systems both in architectural and behavioral aspects. The original AADL is responsible for architectural specification while AADL+ focuses on behavioral description of cyber-physical systems. Furthermore, based on Eclipse platform developed by SEI, we develop a plug-in to OSATE. Users can model and simulate cyber-physical systems on OSATE with the plug-in.

The rest of this paper is organized as follows: Section 2 provides a motivated example. Section 3 defines the general formal model of CPS in AADL+. Section 4 describes the modeling approach of cyber component and physical component in AADL+. Section 5 describes the modeling approach of interactive component in AADL+. Section 6 introduces our AADL+ supporting tool. Section 7 presents the case study on the lunar rover control system. In this section we show how to apply the proposed AADL+ and conduct the simulation.

2 A Motivating example

The landing process is the first step in exploring the moon. As the lunar topography is complex, landing onto the moon requires a safe landing area. Once the landing area is determined, the whole descent stage is independent of the probe. When the lunar probe lands on the lunar surface, the landing gear hits the moon surface with great thrust, which could cause the probe to turn over. Therefore, it is important to ensure the safety of landing.

The lunar rover control system is generally equipped with a radar, a power supply device, a thrust chamber, a controller, deflectors, detectors and landing gears. The functions of radar include sending wireless signal to control center on Earth through antenna and receiving signal from Earth. Power supply device is used to provide power to the lunar rover, especially the solar panel is closed when the process of landing. Thrust chamber is designed to change the thrust constantly according to the velocity and altitude of the lunar rover measured by detectors. Deflectors are used to adjust the altitude gap of landing gear.

During the landing process, the lunar rover has the intrinsic parameters such as force of gravity and mass. Also, velocity, acceleration, thrust force and altitude from lunar rover to the ground is changing continually. If we ignore the loss rate of mass, as shown in Fig. 2 , gravity and thrust force plays an important role on the lunar rover.

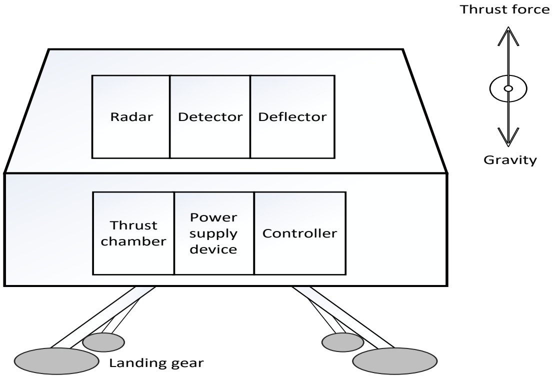


Fig. 2 Structure and force analysis of lunar rover

The relation among properties during the process of landing onto the moon is described in Eq. 1 .

$$\begin{aligned} altitude^{'} &= velocity \\ velocity^{'} &= acceleration \\ acceleration &= \frac{thrustForce - mass \cdot g}{mass} \\ mass^{'} &= abs(thrustForce) \cdot (-lossRate) \end{aligned} \tag{1}$$

Actually, in order to ensure the security of the landing of the lunar rover, there will be four states in the process of falling. In initial state, the lunar rover orbits the moon at an altitude of 54389 m. The velocity is -1894 km/h and the mass is 1038.358 kg. Ideally, the landing process is treated as a straight line. When the track of the lunar rover has been changed, it will prepare to land in this point. That is to say, the event of the landing is triggered, and then the lunar rover starts the first state. Until the falling time of lunar rover is 41.8 second, the value of thrust force keeps 35280N in the first state. In this point, the inner guard holds, and then the system enters the second state. If the falling time is between 41.8 seconds and 202 seconds, the value of thrust force is equal to 1256N in this state. In the point, the falling time is 202 seconds. Another guard holds, and the system action is changed, the lunar rover enters the next state. After this point, the falling time is larger than 202 seconds, and then there is no thrust force any more until the lunar rover lands on the ground.

The lunar rover control system is a typical cyber-physical system. The behavior of the lunar rover control system changes continuously and it is a challenge to describe cyber-physical interaction. More concretely, when the lunar rover reaches the critical point, the system captures the position of the lunar rover. The arrival of the event triggered the operation of the power unit, resulting in a change of force. In critical point, the change in the force of the lunar rover caused the acceleration to change. This process have led to changes in the velocity, altitude, and mass of the lunar rover, that is, changes in the trajectory or behavior of the lunar rover. In the landing process, firstly, the altitude, mass and velocity of the lunar rover changes constantly, that is, the physical environment varies continuously. And then the variables and input value to cyber component is changing continuously. Besides, the communication of interactive component between cyber component and physical component increases the difficulty in modeling the process.

AADL has good expressiveness for the system architecture because of the abundant components. Through defining the properties of these components, the non-functional property of the system can be analyzed. The AADL model is discrete because the model is described in cyber world. AADL without extensions does not support the modeling of physical process because it cannot model continuous dynamics. While AADL+ that we proposed supports modeling of cyber-physical systems, i.e., it supports modeling not only discrete events but also continuous dynamics and cyber-phycical interaction. Through extending the core language of AADL, the AADL+ sublanguage is able to model physical process.

Obviously, it is beyond the ability of AADL to model the process. The motivating example is used to illustrate the insufficiency of AADL to model cyber-physical systems. Based on the idea, we propose an approach to model continuous dynamics and cyber-physical interaction and develop the corresponding tool on the basis of OSATE.

3 General Formal Models of CPS in AADL+

The interaction of information and strategy decision is implemented through 3C (Computation, Communication and Control) under the action of physical layer and cyber layer. In this section, the general formal model of AADL+ is given for modeling cyber-physical systems from the respect of control theory. We present the stochastic differential equation to describe the behavior of cyber-physical systems. It is essential to prove the consistency of the semantics of differential equation with strong Markov property.

To define the system state space, we introduce a relevant parameter d, and then the state space is defined as a subset X of the Euclidean space of dimension d. Within a set Y of regions that are formally called modes and considered to be a category of topological space, the system will evolve. Additionally, white noise is a random signal with a constant power spectral density. In system equation, the noise is modeled by a variable W_t, so the system varies with the change of the variable. White noise is used to model random perturbations (i.e. Wiener process $(W_t, t \leq 0)$ is defined in every region).

Designed behavior refers to the system dynamics in each location which we describe using deterministic differential equations. System behavior would be obviously influenced by white noise. So taking the fact into consideration, the system physical dynamics should be described using stochastic differential equation in Eq. 2:

$$dx(t) = \phi^y(x(t))dt + \varphi^y(x(t))dW_t \tag{2}$$

We use stochastic differential equation to model the behavior that the system is changing continuously. So, it is indispensable to prove that the semantics of the equation satisfies *strong Markov property*. That is to say, the solution $x(t)$ of stochastic differential equation should satisfy the important *Markov property*: The future behavior of the process given what has happened up to time t is the same as the behavior obtained when starting the process at $x(t)$. In stochastic analysis, itô diffusion is a solution to stochastic differential equation. It is easy to prove that itô diffusion satisfies *Markov property* [12]. The term *strong Markov property* is similar to *Markov property*, except that the meaning of *present* is defined in terms of a random variable known as a stopping time (or Markov time). Both the terms *Markov property* and *strong Markov property* have been used in connection with a particular *memoryless* property of the exponential distribution. Similarity, we can also prove *strong Markov property* will hold.

4 Modeling Cyber System and Physical Process Using AADL+

This section focuses on the cyber system and physical process modeling of AADL+. First, we provide the framework of AADL+

4.1 The framework of AADL+

In the framework of AADL+ shown in Fig. 3, the modeling elements include discrete behavior, continuous behavior and component interactive behavior. The discrete behavior is modeled by AADL core component. We describe the continuous behavior (i.e., the physical environment) using variable declaration and equation clause. And the interactive behavior is modeled by the unintentional and intentional interaction, which will be present in Sec. 5.

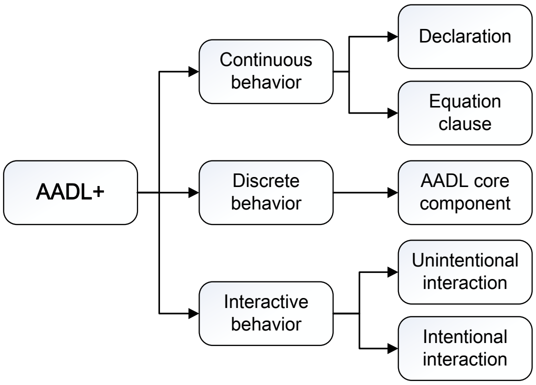


Fig. 3 The framework of AADL+

4.2 Modeling Cyber System Using AADL+

AADL core language is able to model the architecture of cyber part. To model the behavior of cyber component, it is alternative to incorporate BA, HA or BLESS. AADL is a modeling language that provides a standard for the design and implementation of embedded system, which also supports the analysis and design of system architecture with regard to performance-critical properties.

AADL describes system architecture of software and hardware mainly through the concept of components. A component is described by its identity, some properties, interaction with other components or subcomponents, even including implementation and inheritance of the component type. There are three kinds of components: execution platform component, the application software component and the composite component. The cyber components include: process, thread, thread group, subprogram, data, memory, bus, processor and system.

4.3 Modeling Physical Process Using AADL+

Generally, we model physical process by means of differential equations. Actually, the tight coupling between cyber component and physical world is the critical problem for the study of CPS modeling.

This section includes the introduction of predefined data types and properties, and the definition of syntax and semantics of AADL+ for modeling cyber-physical systems.

4.3.1 Predefined Data Types and Properties

The predefined data types and properties are declared in the annex library which makes it extensible for the core language concepts and syntax of AADL language. Within AADL standard, annex library must be declared in a **package** which is used to provide a distinct namespace for declarations and manage the declarations of component type, implementation and property associations. Code Listing 1 is the predefined data types declared within a **package**:

Code Listing 1 Predefined Data Types

```
package AADL+_Annex
  public
  data  Integer
   properties
     Source_Data_Size=>32 bits; # Data  size  of  Integer
      Start =>0;    #  Default  value
     Min=>−Inf;  # Minimum value Inf  represents   a   large   value
     Max=>+Inf;  # Maximum value
  end  Integer ;

  data  Boolean
   properties
     Source_Data_Size=>8 bits;
      Start =>false;
  end  Boolean;

  data   String
   properties
      Start =>false;
  end  String ;

  data  Real
   properties
     Source_Data_Size=>64 bits;
      Start =>0;
     Min=>−Inf;
     Max=>+Inf;
     Unit=>""  # Unit used  in  equations
     DisplayUnit=>"" # Default  display   unit
  end  Real;

  property   set  AADL+_Annex_Properties is
     Cardinality   :   aadlinteger   applies   to  ( data);
      Cardinalities :   list   of   aadlinteger   applies   to  ( data);
  end  AADL+_Annex_Properties;
```

As is seen from the above, we use the AADL **data** component to represent some simple data types such as Integer, String and so on. For example, we create a **data** type Integer for all integers. The **property** association of Integer sequentially declares that the size of the **data** type is 32 bits. The initial value is 0, and minimum value is -Inf, while maximum value is +Inf. If users want to declare a variable of integer type of AADL+ annex, it is supposed to be written like $variableA\ :\ AADL + \_Annex\ ::\ Integer$; where $AADL + \_Annex$ denotes the **package** name. Likewise, the composite data types can be defined by user by means of **data** hierarchies.

Noticing that there are no array types in AADL, we decide to use the concept of cardinality. Cardinality is declared in the **property set** $AADL + \_Annex\_Properties$. So we can declare an integral array of 5 elements like $tab\ :\ dataAADL + \_Annex\ ::\ Integer\{AADL + \_Annex\_Properties :: Cardinality => 5; \};$.

4.3.2 Syntax of AADL+ for cyber-physical systems Modeling

The syntax of AADL+ for modeling cyber-physical systems can be described as follows. There are two optional specifications at the top of the syntactic hierarchy: component clause and equation clause.

```
annex cps_specification {**
  variableDeclaration *
  interactionDeclaration *
  equation *
**}
```

In AADL+, annex is optional. A component without annex (behavior specification) has architecture specifications only. The empty **annex** means there is no physical and interactive specifications in the model.

4.3.2.1 Component Clause

Component clause includes variableDeclaration clause and interactionDeclaration clause. The variableDeclaration clause denotes a declaration of a type component or an array of component. The interactionDeclaration clause denotes the interaction bettween cyber and physical component. Its partial main syntax is shown in Code Listing 2:

Code Listing 2 Component Clause

```
 variableDeclaration   ::=
type_prefix   type_specifier   [ array_subscripts ]
component_list

 type_prefix   ::=
[FLOW|INPUT|OUTPUT][DISCRETE|CONSTANT|PARAMETER]

 type_specifier   ::=
name

component_list  ::=
component_declaration   { "," component_declaration }

component_declaration  ::=
 declaration  comment

 declaration  ::=
IDENT [ array_subscripts ]  [ modification ]

 interactionDeclaration   ::=
Unintentional   Interaction    |     Intentional    Interaction

Unintentional   Interaction    ::=
[RegionOfImpact] |  [ AggregateEffect ]

 Intentional   Interaction    ::=
[ RegionOfInterest ]
```

There are six type prefix: **flow**, **input**, **output**, **discrete**, **constant**, **parameter**, which are only applicable to type and connector of components. But the first three type prefixes are applicable to the elements of component array except that. The remaining type prefixes are called variability prefixes. A **flow** variable, which is a subtype of **Real**, is used to declare a variable when a continuous event happens. A **discrete** variable means that an event happens in discrete time. A **constant** variable never be changed by the external environment once it is assigned some value. While a **parameter** variable cannot be changed only during the simulation process.

4.3.2.2 Equation Clause

The equation clause represents the behaviors of a system, and describes constraints and relations of a model in the system through clause statements.

An equation clause consists of simple expression, i.e. **if** equation, **for** equation, **connect** equation, **when** equation, and **assert** equation. Its syntax is shown in Code Listing 3:

Code Listing 3 Equation Clause

```
equation  ::=
EQUATION { equation_clause";" }

 equation_clause  ::=
(  simple_expression  EQ expression
|  ifEquation  |  forEquation  |  connectEquation
|  whenEquation |  assertEquation   )
comment";"

 ifEquation  ::=
IF  expression  THEN{ equation_clause";" }
[  ELSEIF expression THEN { equation_clause";" } ]
[  ELSE{ equation_clause";" }]
END IF

 forEquation  ::=
FOR IDENT IN expression LOOP{ equation_clause";" }
END FOR

whenEquation ::=
WHEN expression THEN { equation_clause";" }
END WHEN;

connectEquation  ::=
CONNECT "(" connector "," connector ")"

 connector  ::=
IDENT [  array_subscripts   ][  "." IDENT [ array_subscripts  ]  ]

 assertEquation  ::=
ASSERT "(" expression "," STRING {"+" STRING }")"
TERMINATE "(" STRING { "+" STRING } ")"
END ASSERT
```

4.3.3 Semantics of AADL+ for Cyber-physical System Modeling

This section is an outline of the semantics of AADL+ for modeling cyber-physical systems. In this paper, we provide formal semantics to describe the semantics of partial clauses in AADL+. Formal semantics describing the system behaviors is composed of some rules. The rule looks like as follows:

$$\frac{t_1 \Rightarrow p_1 \quad \wedge \quad t_2 \Rightarrow p_2 \quad \wedge \quad \ldots \quad \wedge \quad t_n \Rightarrow p_n}{t \Rightarrow p}$$

Each clause is called a sequence, and the sequence is definitely divided to two parts: above the line or under the line. All the sequences of the former are called premises, and the rule of the latter is called conclusion. Sometimes, there may be the composition of many premises and conclusions in a rule. Only if each of the sequences in the premises is true, the conclusion will hold.

There are two subsections discussing about the semantics of declaration and equation clause.

4.3.3.1 Declaration

This subsection provides semantics of declaration of component clause.

Declaration of Component Clause Type prefixes such as **flow**, **input**, **output**, **discrete**, **parameter** and **constant** are only applied to type and connector of components. Type prefixes **flow**, **input** and **output** are not only characterized a structured component, but also used for subcomponent members of the component. If no component has the same type prefix, the above three type prefixes can only be used as a structured component [13].

Generally, the type prefixes **discrete**, **parameter** and **constant** are used most often. The type prefix **discrete** is only applied to real variable, indicating that the declared variable is a discrete variable, which has a vanishing time derivative (the derivative of this **discrete** variable with respect to time is 0) and its value is changeable only when an event happens. To the contrary, a continuous-time variable has a non-vanishing time derivative (the derivative of this variable with respect to time is not equal to 0). A real variable within a **when clause** must be a discrete-time variable no matter whether it is prefixed with keyword **discrete**. If a real variable is not within a **when clause** and has not any type prefix, it is a continuous-time variable. The following is a declaration of **discrete** real typed variable:

discrete Real v;

The variable denoted by the type prefix **parameter** is a parameter constant which indicates that it is an unchangeable constant variable in the simulation process. And only before the simulation, the value of this variable can be modified. A real typed variable is declared with prefix **parameter**:

parameter Real v = 20;

The type prefix **constant** denotes that the variable is a named constant. A **constant** variable is unchangeable after it is declared. A string variable with **constant** is declared:

constant String str = "test";

It is worth noting that messages through event and event data port transmit with queuing. Events from the environment trigger the execution of system and an event is responded in one moment. On dispatch, delivery depends on *Dequeue_Protocol*. So it need maintain with a queue. However, variables maintain without a queue because variables interact with cyber component using **input** and **output** and map with data port.

Here is an example of AADL+ annex showing the declaration of various variables.

```
annex cps_specification {**
 constant Real pi = 3.141592653589;
 parameter Real x = 5;
  Real time;
 Real result;
 equation
    result = cos(2*pi*x*time);
**}
```

4.3.3.2 Equation Clause: Modeling Discrete Events and Continuous Dynamics

It mainly includes **if clause**, **for clause** and **when clause** whose semantics is described below and the syntactic part can be seen in section 4.3.2.2. Additionally, the equation clause can access the ports and data subcomponents of the AADL+ component. Most importantly, equation clause provides the capability of modeling discrete events and continuous dynamics for AADL+. First we introduce some built-in operators.

Built-in Operator In equation clause, the invocation of a built-in operator operates the system variables, which is just like a mathematical function call. But the difference is that the result of invoking built-in operator is concerned with not only the arguments like mathematical function call but also the status of the simulation. The following is several common built-in operators.

der(arg) The operator means the derivative of the argument concerning the temporal variable *t*. Obviously, the arg must not be a variable with the prefix discrete because the arg with the type discrete is meaningless. If arg is a constant, as is known to all, the result of **der**(arg) is assigned zero. A second-order differential equation can be described with **der**(**der**(arg)). This operator provides the capability of modeling continuous dynamics for AADL+. Its natural semantics is as follows:

$$\frac{arg \notin D \quad \wedge \quad arg \notin C}{\mathbf{der}(arg) := arg_t'}$$

$$\frac{arg \notin D \quad \wedge \quad arg \in C}{\mathbf{der}(arg) := 0}$$

where D is the set of discrete-time variables, C the set of all constants, arg_t' the derivative of arg concerning time t.

reinit(arg, exp) When an event arrives or a guard condition holds, the operator is defined to reinitialize the argument arg with the expression exp. Meanwhile, it is essential that the argument should be a subtype of Real type and would process after the operator **der**. Its natural semantics is as follows:

$$\frac{arg \in R \ \wedge \ arg \in DER \ \wedge \ e \Rightarrow true \ \wedge \ reinit(arg, exp)}{arg := exp}$$

where R is the set of all **Real** types, DER the set of all the arguments processed by **der** operator, e the event, exp the expression to reinitialize the argument.

If Clause The **if clause** contains Boolean expressions and assignment equations, which is defined to model a system that continuous physical dynamics of the system change after conditional evaluation. Unless the nesting of **if clause**, an **if clause** contains **if subclause**, **else if subclause** and **else subclause**. Among them, there is only one **if subclause** and no more than one **else subclause**, then **else if subclause** is optional. Taking the evaluation of the Boolean expressions for **if clause** into account, there is always a situation that the Boolean expression evaluating to true, then the corresponding assignment equations of **if subclause** or **else if subclause** execute. Unless the conditional expression never be evaluated to true, the **else subclause** would be executed.

Additionally, we have to notice that the equation within an **if clause** is always effective in the whole section where conditional expression evaluates to true. And the corresponding variables are evaluated continuously in this whole section.

The semantics of **if clause** can be seen as follows:

$$\frac{\langle e,\sigma\rangle \Rightarrow true \;\;\wedge\;\; \langle P,\sigma\rangle \xrightarrow{a} \langle P,\sigma_1\rangle}{\langle \textbf{if } e \textbf{ then } P \textbf{ else } Q\ ,\ \sigma\rangle \xrightarrow{a} \langle P,\sigma_1\rangle}$$

$$\frac{\langle e,\sigma\rangle \Rightarrow false \;\;\wedge\;\; \langle Q,\sigma\rangle \xrightarrow{a} \langle Q,\sigma_1\rangle}{\langle \textbf{if } e \textbf{ then } P \textbf{ else } Q\ ,\ \sigma\rangle \xrightarrow{a} \langle Q,\sigma_1\rangle}$$

The rule defines the dynamic semantics of the general binary choice of **if clause** in AADL+ annex, where e is the conditional expression of **if clause**, P and Q respectively the equation of the true choice and the false choice, σ and σ_1 the state before and after the execution of **if clause**. If conditional expression e evaluates to true, then the equation P executes, otherwise the equation Q executes. And the state transforms from σ to σ_1.

When Clause The **when clause** provides the capability of modeling discrete events for cyber-physical systems, which describes the equation applicable only at some discrete time. There is no double that the expression within a **when clause** is a discrete-time Boolean expression. Only when the Boolean expression evaluates to true, the corresponding equation will take effect. What's more, the equation statement is activated only at some time when the corresponding Boolean expression is true, which is different from that of **if clause**. The time is the current time, which starts from the beginning of the simulation of the system and changes with the process of execution. A new event occurs at this time, resulting in the system triggered into next state. Here we set an example to explain the above statements.

```
annex cps_specification {**
     parameter Real x = 4;
     parameter Real v = 1.5;
     Real t;
      Real time;
     Boolean bool;
  equation
     t = v* sin(x* time);
     when t > 0 then
     bool = not pre(bool);#pre(bool) returns
      #the value of bool after the last
      #event iteration at the instant
     end when;
**}
```

Fig. 4 shows the simulation result of the above model.

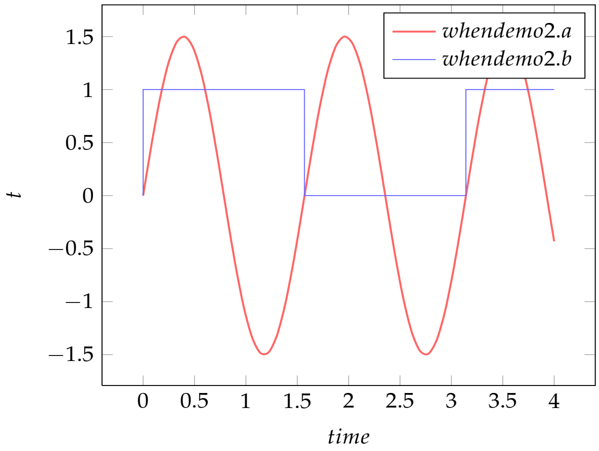


Fig. 4 Simulation result of the above AADL+ annex
Blue curve: Boolean bool Red curve: Real t.

As we can see from Fig. 4, only at the time when the evaluation of expression $t > 0$ changes to true, the equation within **when clause** would be activated, that is the value of Boolean variable *bool* would change against the previous value.

The above model of AADL+ annex is actually identical with the hybrid finite automata (see Fig. 5).

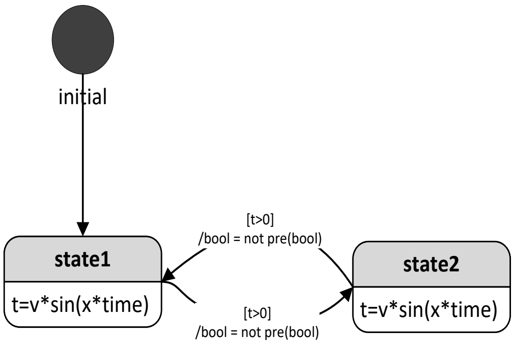


Fig. 5 Hybrid finite automata of the example AADL+ annex.

It is seen that the conditional expression ($t > 0$) of **when clause** corresponds to the guard of a transition in the above state machine. And the equation within the **when clause** ($bool = notpre(bool)$) corresponds to the effect of a transition in the state machine.

The semantics of **when clause** can be seen as follows:

$$\frac{\langle pre(e),\sigma\rangle \Rightarrow false \wedge \langle e,\sigma_1\rangle \Rightarrow true \wedge \langle P,\sigma_1\rangle \xrightarrow{a} \langle P,\sigma_2\rangle}{\langle \textbf{when } e \textbf{ then } P\ ,\ \sigma_1\rangle \xrightarrow{a} \langle P,\sigma_2\rangle}$$

Where $pre(e)$ and e is the evaluation of a boolean expression in previous state σ and current state σ_1, it is clear that the evaluation of $pre(e)$ is false, P a executable equation statement of **when clause** in current state σ_1, σ_2 the state after P executes. The rule describes the dynamic semantics of **when clause** for cyber-physical systems in AADL+ annex, When the value of $pre(e)$ is false in previous state and the value of conditional expression e is true in current state, then the statement P executes and the system enters next state σ_2.

For Clause The **for clause** describes the continuous dynamics of cyber-physical systems, which contains loop-variable, loop-set and loop-body in AADL+ annex. In each loop, loop-variable is like an event to trigger the execution of the system. Taking the state transition of a system into account, once the variable is assigned a value in the loop-set, the corresponding equation will be executed in the loop-body. The semantics of **for clause** is shown as follows:

$$\frac{i \in loopSet \ \wedge \ \langle P\ ,\ \sigma\rangle \xrightarrow{a} \langle P, \sigma_1\rangle}{\langle \textbf{for}\ i\ \textbf{loop}\ P\ ,\ \sigma\rangle \xrightarrow{a} \langle P, \sigma_1\rangle}$$

where $loopSet$ is the loop-set, i the loop-variable in $loopSet$ changed in each loop, P the equation of the loop body in current state σ of the system, σ_1 the state after P executes in the state σ. The rule above describes the dynamic semantics of **for clause** for cyber-physical systems in AADL+ annex. For each loop-variable, there is always a corresponding equation executed. Otherwise, the program skips out the loop, and then the state transfers from σ to σ_1. Also, the equivalent natural semantics of **for clause** can be the following form:

$$\frac{i \in loopSet \ \wedge \ \langle P, \sigma\rangle \xrightarrow{a} \langle P, \sigma_1\rangle}{\langle \textbf{if}\ i\ \textbf{then}\ (P;\ \textbf{for}\ i\ \textbf{loop}\ P)\ \textbf{else}\ skip, \sigma\rangle \xrightarrow{a} \langle P, \sigma_1\rangle}$$

According to the above introduction of AADL+ annex semantics, we have the following conclusion. In AADL+ annex, the built-in operator **der** can be used to express the continuous models (i.e. $\textbf{der}(x) = 2x + 3$). **If clause** and **when clause** can be used to represent discrete models.

5 Modeling Cyber-Physical Interaction Using AADL+

Cyber-physical systems put emphasis on the interaction between cyber components and physical systems. The interactive component is externalized as a device component in AADL [14], which can be regarded as an indispensable part in the systems and be accessed and controlled by the cyber and physical components. In AADL+, sensor and actuator are both the interactive components. Sensors collect relevant data from physical world, while actuators affect the physical world according to computing result. So we can model the closed-loop interaction between cyber and physical components through interactive components.

The continuous signals in physical system can be recognized by sensors in the interactive component and transferred to digital signals in cyber world. Specifically, in a cyber-physical system, the **input** and **output** in physical component are collected by sensors in interactive component, and connect with cyber world through in data port and out data port respectively. Similarly, **discrete** and **flow** are collected by sensors in interactive connector component, and trigger information world through in event port and out event port respectively. The corresponding changes will happen in cyber system.

After that, the digital signals will respond to the physical system by the interactive component. Equation is executed in component implementation. The **equation** clause can access the ports and data subcomponents of the cyber component. It is worth noting that there also are operators such as addition, minus, multiply, division, integral in a concrete equation implementation. Through component implementation, the equations and variables can be inherited from other component implementation.

Inspired by the work of Banerjee A, Kandula S, Mukherjee T, et al. [15], we introduce some concepts to model the interaction of cyber-physical systems. A cyber-physical system has got computing units embedded in a physical environment. The computing unit interacts with its environment in two ways: unintentionally and intentionally.

5.1 Modeling of Unintentional Interaction

Unintentional interaction refers to the side effects on physical world because of computation and communication of computing units (i.e. thermal, electromagnetic). There are two categories of unintentional interactions: region of impact and aggregate side-effect [15].

5.1.1 Region of Impact (ROIM)

Region of impact (ROIM) refers to the spatial extent to which physical world is influenced by computing units. In AADL+, region of impact can be modeled through the following subcomponents: impacting (or impacted) parameters, enclosing region. There are corresponding port groups: cyberToROIM, cyberToimpactingP, cyberToimpactedP and cyberToEnRe. Interactive components connect with cyber components through these port groups. This section will describe how to use AADL+ to model the region of impact through the above several subcomponents in turns. The relevant syntax will be provided in Code Listing 4.

5.1.1.1 Impacting (or Impacted) Parameters

Impacting parameters refer to the physical properties of computing unit causing side effect (i.e. power circuitry, specific absorption rate), while impacted parameters refer to the physical properties of physical environment that are affected. Code Listing 4 is the syntax of modeling impacting (and impacted) parameters using AADL+:

Code Listing 4 Modeling of Unintentional Interaction

```
RegionOfImpact ::=
(IMPACTINGPARAMS LCURLY impactingParams RCURLY)
(IMPACTEDPARAMS LCURLY impactedParams RCURLY)
(ENCLOSINGREGION LCURLY enclosingRegion RCURLY);

impactingParams ::=
(
   (parse_propertyCOMMA) +
|  ( parse_property   )
|  ( parse_data  COMMA)
|  ( parse_data )  +
)  (SEMI);

impactedParams ::=
(
   (  parse_property  COMMA ) +
|  (  parse_property   )
|  (  parse_data  COMMA )
|  (  parse_data   )  +
)  (  SEMI );

enclosingRegion  ::=
(
   (LENGTH '=' INT SEMI)(WIDTH '=' INT SEMI)
|   stat   ) ;

AggregateEffect  ::=
(AGGREGATEEFFECT LCURLY
( function  SEMI) +RCURLY );

function   ::=
(LPAREN (parse_property |  parse_data )
COMMA (PLUS | MINUS) RPAREN);
```

The following is an AADL+ model of Body Area Network [16] which is a typical cyber-physical system. One of the features of Body Area Network is to measure the human body temperature. But the power circuitry of computing units would generate amounts of heat which may have a side effect on the calculation of skin temperature. So it turns out that, in this system, power circuitry is an impacting parameter, while skin temperature is an impacted parameter.

```
annex cps_annex {**
  RegionOfImpact {
    ImpactingParameter {
      Value(PhysicalPropertySet::
                PowerCircuitry);
    }
    ImpactedParameter {
      Temperature;
    }
    EnclosingRegion {
      Length = 28;
      Width =  23;
    }
  ...
**}
```

5.1.1.2 Enclosing Region

Enclosing region can be regarded as an algebraic equation to denote spatial extent of ROIM. To be specific, we can 1) use length and width to parameters to model rectangle boundaries; 2) use an algebraic equation to model circle or ellipse; 3) or use thresholds (which are limits on variation of impacted parameters) to model the region boundaries that are irregular and unable to be modeled by means of simple geometric equations (i.e. $Temperature < 256K$). The syntax of modeling enclosing region using AADL+ is shown in Code Listing 4:

5.1.2 Aggregate Side-effect

Aggregate side-effect is a combined side effect of computing units when their region of impact (ROIM) is an overlap. Usually, aggregate side-effect is a function of impacting parameters. For example, in Body Area Network, there are two sensors that are responsible for retrieving the data about the human body temperature. As is seen from Fig. 6, both of the sensors have thermal side-effects (which would affect the computation of skin temperature) and respective region of impact. The overlapping part is the region of aggregate side-effect which is colored red in this figure.

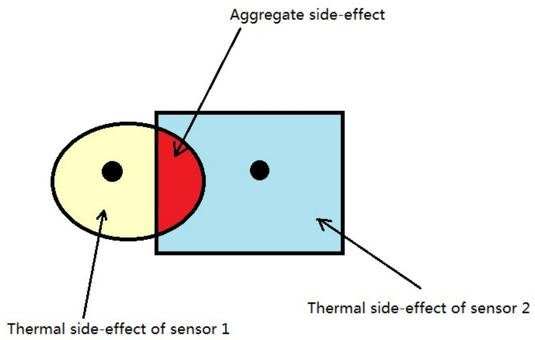

Aggregate side-effect
Thermal side-effect of sensor 2
Thermal side-effect of sensor 1

Fig. 6 Aggregate thermal side-effect of two sensors.

Aggregate side-effect results from the overlap of multiple regions of impact. Some impacting parameters have enhanced side-effect on the computation of some properties, while others have weakening side-effect on it. In the example of Fig. 6, both of the impacting parameters (heat generated by power circuitry of sensor) from two sensors have definitely effect on measuring skin temperature, because the heat generated by power circuitry will enhance the final computation value of the skin temperature. In AADL+, '+' and '-' are respectively used to denote whether the impacting parameter has enhanced or weakened side-effects on the computation of properties. Code Listing 4 gives the syntax of modeling aggregate side-effect.

The below is the corresponding AADL+ model of the example showed in Fig. 6.

```
annex cps_annex {**
  AggregateEffect {
   (value(Sensor1PropertySet::PowerCircuitry),
    '+');
   (value(Sensor2PropertySet::PowerCircuitry),
    '+');
  }
**}
```

5.2 Modeling of Intentional Interaction

Other than unintentional interaction, the computing unit certainly interacts with its environment intentionally for the execution of system operations. Because the modeling approach of intentional interaction is similar to that of unintentional interaction in many aspects, we just briefly introduce the exclusive part of intentional interaction. Intentional interactions are characterized by region of interest.

Region of Interest can be regarded as a counterpart of region of impact which is a characteristic in unintentional interaction. Region of interest bounds the space in which variation of monitored parameters is considerable, which is communication region of wireless sensor network when collecting data from physical environment. Compared with region of impact, region of interest can be modeled through monitored parameters and enclosing region. Monitored parameters refer to the application parameters that we want to monitor ranges such as sensing range or communication range. Enclosing region is actually an algebraic equation to denote the spatial extent of region of interest. The port groups connecting with cyber component are cyberToROIN, cyberToMP and cyberToER. Code Listing 5 is the syntax of modeling intentional interaction using AADL+:

Code Listing 5 Modeling of Intentional Interaction

```
 RegionOfInterest  ::=
(MONITOREDPARAMS LCURLY monitoredParams
RCURLY)
(ENCLOSINGREGION LCURLY enclosingRegion
RCURLY) ;

monitoredParams ::=
(
( parse_property  COMMA)+
|  ( parse_property  )
|  ( parse_data   COMMA)
|  ( parse_data )+
)  (SEMI) ;

enclosingRegion  ::=
((LENGTH '=' INT SEMI)
(WIDTH '=' INT SEMI)
|   stat   );
```

Last but not least, intentional and unintentional interactions depend on the state of computing units. For instance, in Body Area Network, there are two sensors both of which have thermal side-effect. Sensor1 is operating at low frequency, while sensor2 at high frequency. It turns out that sensor2 has more thermal side-effect than sensor1. Thus, we can say that region of impact varies with operating mode of a sensor. The following is a simple AADL+ model for the above situation.

```
  system sensor
  end sensor;
  system implementation sensor.ins1
  modes:
     Lowpower: initial mode
     HighPower: mode
     ...
  annex cps_annex {**
    ComputingState (Lowpower) {
     RegionOfInterest{
       ...
     }
     RegionOfImpact {
       ...
     }
    ComputingState (Highpower) {
     RegionOfInterest{
       ...
     }
     RegionOfImpact {
       ...
     }
   **}
end sensor.ins1
```

6 AADL+ Supporting Tool: A Plug-in to OSATE

In this section we describe our AADL+ supporting tool that is a plug-in to OSATE.

6.1 OSATE tool

OSATE is a best-known open-source tool for modeling AADL, which can be downloaded in . This tool is appropriate for both end users and tool developers, because it not only offers textual and graphical editors to model AADL and some analysis tools for end users, but also supports AADL meta-model for tool developers. In OSATE, AADL model is presented in three forms: AADL text, XML (Extensible Markup Language) and graphical AADL editor. The above three forms are totally equivalent in semantics. OSATE development platform can organize the whole model as system architecture by the function of instantiation, and organize the components of all categories (including processor, process, thread, bus and so on) in the system as a whole according to hierarchy relationship.

So far, OSATE has been plugged with a set of plug-ins, including error modeling analysis, resource budget and allocation analysis, security level checking, MetaH generation [6], error model annex and so on.

6.2 Features of AADL+ Supporting Tool

The AADL+ supporting tool that we developed is a plug-in to OSATE. Its main features are lexical and syntax analysis, syntax checking, schedulability analysis and simulation for AADL+ models. Simulation, as an approach to ensure the correctness of the design, can reduce design costs and time of cyber-physical system. The features are shown in Fig. 7.

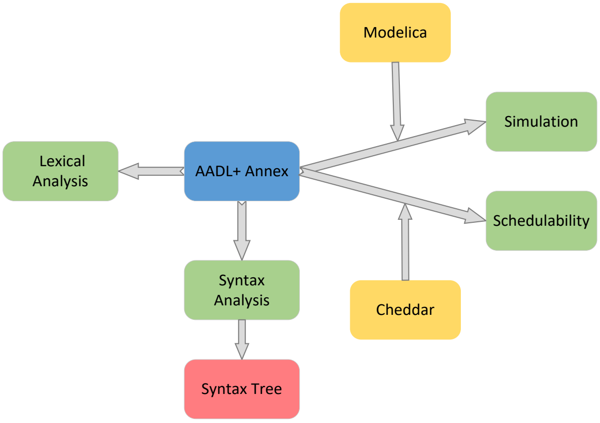


Fig. 7 The features of the tool.

When users take lexical and syntax analysis, several options can be adopted: a) only do lexical analysis; b) redirect error information to standard output; c) generate abstract syntax tree (Fig. 8); d) only return an exit code, without any outputs. During syntax check, if there are some errors for the AADL+ models, then our tool would display the line number where each error is located. After confirming the syntax of AADL+ models correct, we can set some parameters and then simulate the models. The simulation result would be displayed in the form of graph.

When checking the syntax of AADL+ annex, the plug-in will prompt success if the syntax is totally correct. Otherwise, the plug-in will prompt where syntax errors occur.

The user interface of the AADL+ supporting tool contains the three parts: textual editor (where users build AADL+ models), error reporting console (where errors of models are displayed) and the frame of syntax check and analysis (where user can check syntax and see the generated abstract syntax tree).

In order to ensure the correctness of the modeled cyber-physical system, we implement syntax checking and simulation in the paper. While a user is modeling a cyber-physical system, if there is an error, the plug-in will return the position of the error with Red Cross so that he can check the model and modify the corresponding error. So, syntax checking will ensure the correctness of lexis and syntax.

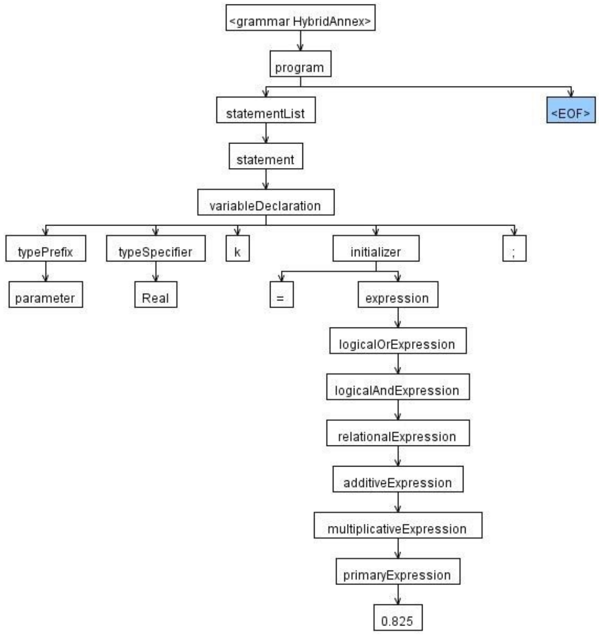


Fig. 8 Abstract syntax tree.

AADL+ is an extension of AADL and it is used to design cyber-physical systems. In order to ensure safety and reliability, we schedule the model to present the use of time and resources of CPU. Clearly, it is impossible to schedule the physical environment and interactive part of a system. So, the cyber part of the model, i.e. AADL core components and properties, is scheduled.

AADL+ is able to simulate the model of cyber-physical systems, while AADL can't. As for simulation, we define the mapping rules to translate AADL+ model to modelica model [13]. It will generate an intermediate file and the intermediate file is linked to modelica simulator. The model is simulated through modelica simulator, which is described in Appendix A in detail. Simulation is used to evaluate the correctness of the system with continuous behavior. Simulation result demonstrates the changing process of the system along with time.

7 Modeling and Analyzing Lunar Rover Control System

7.1 AADL+ Model of the Lunar Rover

In the model, we focus on the landing process of lunar rover. In the landing process, detectors can be seen as sensors, which is used to gain the altitude from physical environment. And deflectors serve as actuators to control the altitude gap of landing gear. Thrust chamber, radar and controller are modeled as threads, while power supply device is modeled as device connecting with processor and memory via bus. Fig. 9 shows the model.

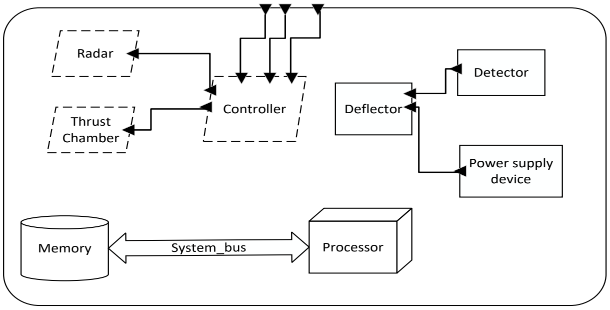


Fig. 9 The model of the lunar rover control system.

In this model, detectors get the altitude of the lunar rover, while the lunar rover adapts itself to make a safe landing on the moon through deflectors. Power supply device provides energy for the lunar rover. For instance, when the detectors take a picture of the ground and analyze the altitude of the lunar rover, it needs energy from power supply device. The controller gets data from the system. For one thing, the controller controls the radar to send the information to the antenna. For another, it sends out signals to thrust chamber to change the thrust. It is essential for the thread to be bound to the processor.

For simplicity, we just provide the partial AADL models that contain the AADL+ annex. The following is the AADL+ model of controller.

```
thread controller
features
  originalMass: in data port;
  originalAltitude: in data port;
  originalVelocity: in data port;
  controlRadar: out data port;
  controlThrust: out data port;
end controller;

thread implementation controller.impl
properties
  Initialize_Execution_Time => 15s..25s;
  Initialize_Deadline => 32s;
  Compute_Execution_Time => 230s..260s;
  Compute_Deadline => 300s;
  annex cps_specification {**
   Real mass(start = originalMass);
   Real gravity;  #gravity forcefield
   Real altitude(start = originalAltitude);
   Real velocity(start = originalVelocity);
    Real time;
   Real acceleration;
   parameter Real G = 6.672e-11;
   parameter Real moonMass = 7.35e22;
   parameter Real radius = 1.738e6;
   parameter Real lossRate = 0.000304;
     #loss rate of mass
   equation
     der(altitude) = velocity;
     der(velocity) = acceleration;
     thrustForce - mass*gravity
          = acceleration*mass;
     der(mass)
          = -lossRate*abs(thrustForce);
     gravity
          = G * moonMass/(altitude + radius)^2;

   parameter Real firstPeriod = 41.8;
   parameter Real secondPeriod = 202;
   parameter Real firstForce = 35280;
   parameter Real secondForce = 1256;
        Real thrustForce; #Thrust force
   equation
     thrustForce = if(time < firstPeriod)
     then firstForce
     else if(time < secondPeriod)
     then secondForce else 0;
   **}
end controller.impl;
```

Fig. 10 shows a state machine model corresponding to the above AADL model with AADL+ annex. We draw the figure using UML state chart to present an intuitive impression of the landing process.

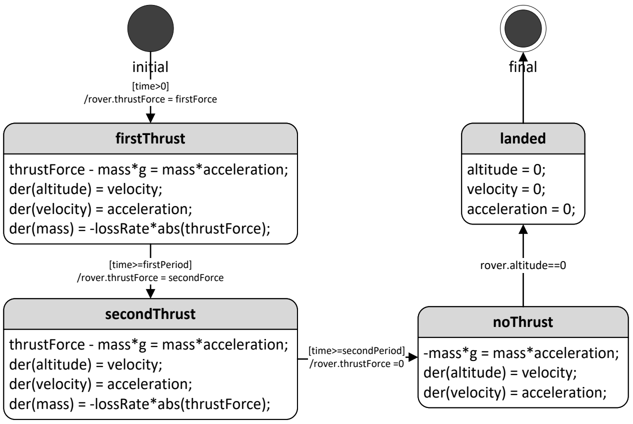

initial
[time>0]
/rover.thrustForce = firstForce
firstThrust
thrustForce - mass*g = mass*acceleration;
der(altitude) = velocity;
der(velocity) = acceleration;
der(mass) = -lossRate*abs(thrustForce);
[time>=firstPeriod]
/rover.thrustForce = secondForce
secondThrust
thrustForce - mass*g = mass*acceleration;
der(altitude) = velocity;
der(velocity) = acceleration;
der(mass) = -lossRate*abs(thrustForce);
[time>=secondPeriod]
/rover.thrustForce =0
noThrust
-mass*g = mass*acceleration;
der(altitude) = velocity;
der(velocity) = acceleration;
rover.altitude==0
landed
altitude = 0;
velocity = 0;
acceleration = 0;
final

Fig. 10 State machine of lunar rover model.

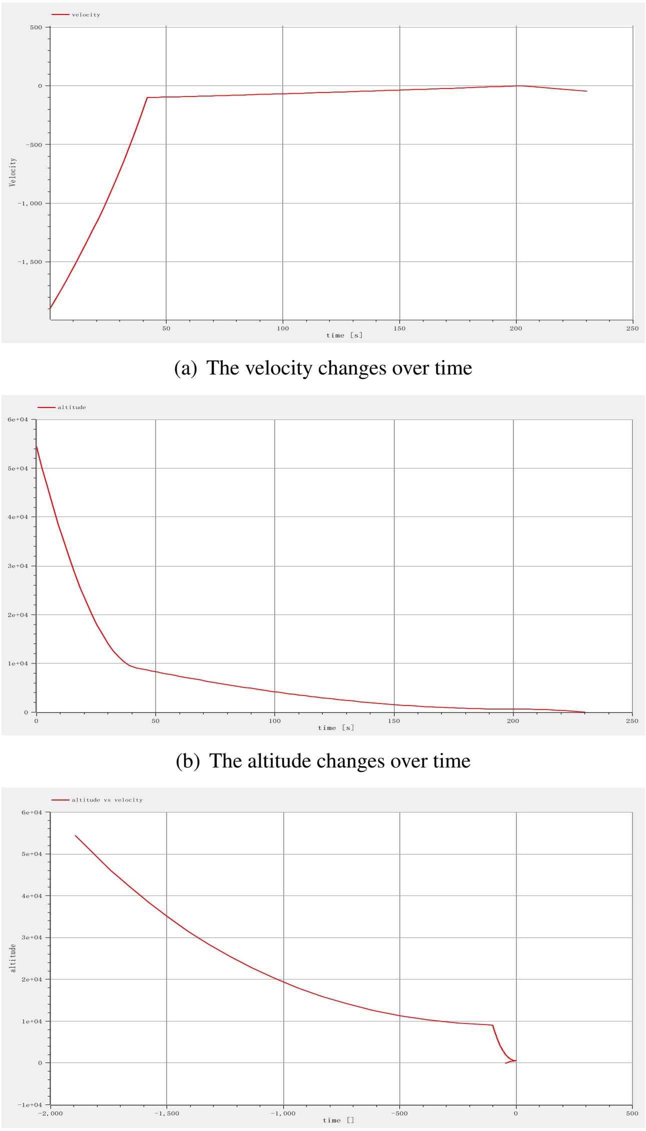
We can see clearly on the above state machine that there are four states in this model. In each state, lunar rover must control the thrust force to change the acceleration. When the velocity reduces continually and is close to zero, it will land onto the moon successfully. Every transition is triggered by the falling time of the rover, that is, each state corresponds to only one period of landing process.

7.2 Simulation Result

Then, we analyze syntax and simulate the model of the landing process. We do simulation of the AADL+ model through linking with Modelica. Fig. 11 and Fig. 12 show the simulation result.

The mapping rules can be seen in Appendix. After syntax checking, it generates an intermediate file that can be identified by Modelica. When we click *Annex simulation* in the menu, it will call OpenModelica. Then we simulate the model through executing the intermediate file in Modelica.

Fig. 11 shows the changes in velocity and altitude. Fig. 11(a) presents the variation of velocity. As the acceleration changes caused by a change in thrust, the increase in velocity at 41.8s slows down, and the velocity is equal to 0 at 202s. After 202s, the lunar rover accelerated motion until landing on the lunar surface. Fig. 11(b) presents the change of altitude is similar to that of velocity. the lunar rover finally lands on the moon when the altitude is equal to 0. Fig. 11(c) shows the altitude changes as the velocity changes.

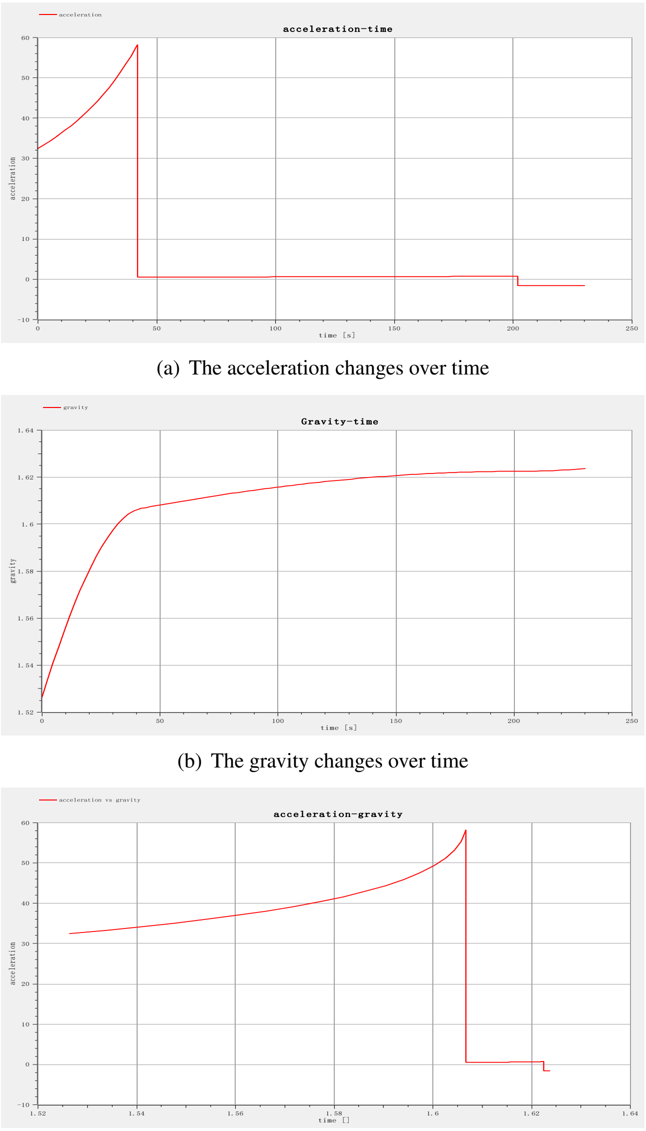


(a) The velocity changes over time

(b) The altitude changes over time

(c) The altitude changes over the velocity

Fig. 11 Simulation of altitude and velocity



(a) The acceleration changes over time

(b) The gravity changes over time

(c) The acceleration changes over the gravity

Fig. 12 Simulation of acceleration and gravity

Fig. 12(a) presents the acceleration varies with the time. According to Eq. 1, the change in acceleration is caused by a change in force. Fig. 12(b) presents the gravity varies with the time. According to Eq. 1, the gravity increases as the height decreases. Fig. 12(c) shows the relationship between acceleration and gravity.

By the impact of the initial velocity, mass, altitude and the initial thrust, the system operates in continuous dynamic environment. The velocity increases at the beginning of the landing process. At 41.8s and 202s, the triggering of time at different phase, makes the change of thrust. The change in thrust causes a change in acceleration, resulting in changes in velocity, altitude, and mass. Of course, if the time and thrust of each phase are reset, the initial velocity, altitude and mass will also make the corresponding changes.

7.3 Schedulability analysis

AADL provides abundant components to model the architecture of a system. When modeling the system, a lot of properties can be selected to every component. So, AADL has the advantage of non-functional analysis. There are a lot of tools for analyzing the system model with AADL. But as for AADL+, we schedule the model through linking to Cheddar [17]. Cheddar is a free real time scheduling tool and it is designed for checking task temporal constraints and buffer sizes of a real time application/system. It can also help us for quick prototyping of real time schedulers. The XML file is used for Cheddar as the input. So the main work is getting the XML file.

Specifically, firstly, we model the cyber-physical systems using AADL+. After checking syntax of AADL+ model, we translate AADL+ model to Cheddar. And then we analyze the model by the way of Cheddar.

7.4 AADL+ vs. Charon Annex

Charon Annex, which bases on the notions of agent and mode, is a sublanguage of AADL for specification of hybrid systems [18, 19]. Through this sublanguage, AADL is applicative to model hybrid systems. In Charon Annex, individual components are described as agents. Charon Annex provides agents with the operations of composition, instantiation and hiding. Individual behaviors of agents are described as modes. Charon Annex provides the operations of encapsulation, instantiation and scoping for modes. More importantly, Charon Annex supports discrete and continuous behaviors.

Now we provide the partial Charon Annex models for the landing process of lunar rover as follows:

```
thread implementation landing_process.charon
annex Charon {**
  mode firstThrust = brake_one(...);
  mode secondThrust = brake_two(...);
  mode noThrust = free_fall(...);
  mode landed = land_on_moon(...);
  trans from firstThrust to secondThrust do {
    rover.thrustForce = secondForce;
  }
  trans from secondThrust to noThrust do {
    rover.thrustForce = 0;
  }
  trans from noThrust to landed do {
    rover.altitude = 0;
    rover.velocity = 0;
    rover.acceleration = 0;
  }
  mode brake_one(...) {
    diff diff_brake_one {
         d(rover.altitude)
             = rover.velocity;
    d(rover.velocity)
             = rover.acceleration;
    d(rover.mass)
             = -lossRate*abs(rover.thrustForce);
    }
  ...
  **}
end landing_process.charon;
```

We see that Charon Annex is also able to handle hybrid models in AADL. But the approach adopted by AADL+ and Charon Annex for modeling the basic hybrid system is different, which is showed in Tab. 1. The second row shows that the language modifiers are defined to indicate the type of a variable for these two description methods. The third row shows the relation between a dynamical system and a specific state. AADL+ provides various equation sets depending on the events. In Charon Annex, a mode can be described by algebraic and differential equations. The last row shows the interaction between the discrete and continuous processes. Both AADL+ and Charon Annex have specific modifiers to distinguish discrete and continuous signals.

Table 1 Comparison of AADL+ and Charon Annex for modeling hybrid system

Name	AADL+	Charon Annex
Discrete /Continuous specification	defined by language modifier	defined by language modifier
Discrete /Continuous interface	differential equation set depending on event	mode refinenement into continuous dynamics
State/Dynamics mapping	indirect (**when** statement)	indirect

In the aspect of simulation, we only can use the Charon simulator to simulate the above Charon Annex models out-side Osate environment because the simulator of Charon Annex is not integrated into AADL. The running time of simulation and simulation results of the above Charon Annex model are nearly identical with those of AADL+ model. But there are still some different metrics (i.e. CPU usage) between AADL+ simulator and Charon simulator. We run our analysis on a laptop computer with an Intel-i5 CPU and 8GB of RAM. And then we measure CPU usage and acquire the value of CPU usage using AADL+ simulator with aid of modelica and Charon simulator for 300 times after executing the simulation. The execution time keeps invariant in the same environment. Then we value the average CPU usage per second during the simulation. Fig. 13 shows the average CPU usages of computer while AADL+ and Charon simulator are simulating the models in case study. The average cpu usage during the simulation by AADL+ simulator is about 92.3667%, which is a little bit lower than that by Charon simulator: 94.9501%.

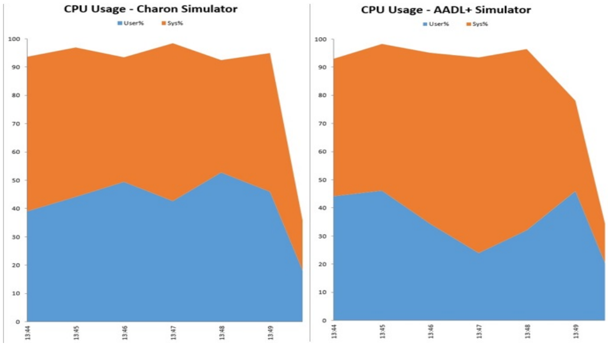


Fig. 13 CPU usage during simulation by AADL+ and Charon simulator.

Most importantly, Charon Annex only focuses on the modeling for hybrid systems, does not contain the feature of modeling the interaction between cyber components and physical process as AADL+ does. It has the ability to model the physical component with differential equations. But it isn't enough to model the interaction between cyber component and physical component because there is no corresponding port to deliver messages. The Body Area Network in Sec.5 is a typical example, but Charon can't support to model it. Also, there is still no corresponding plug-in for Charon Annex, which limits its use. Hence, the advantages of AADL+ over Charon Annex are:

1. **totally integration into AADL as a sublanguage;**
2. **being able to build CPS models;**
3. **simple syntax for modeling continuous dynamics and discrete controls;**
4. **modeling interaction between cyber components and physical components;**
5. **as a plug-in to OSATE.**

Tab. 2 presents the overall comparison between AADL+ and Charon Annex.

Table 2 AADL+ vs. Charon Annex

	AADL+	Charon Annex
Model Hybrid Systems	√	√
Build CPS Models	√	×
Model Cyber- Physical Interaction	√	×
Integration into AADL	√	×
Toolset	√	×

7.5 AADL+ vs. AADL

AADL is a language designing for architecture of embedded systems. It supports an architecture-centric, model-based development approach throughout the system life cycle. AADL describes system architecture of software and hardware mainly through the concept of components. A component is described by its identity, some properties, interaction with other components or subcomponents. With those properties, it can be used to analyze its non-functional property. However, AADL has some disadvantage. For one thing, AADL can't describe the continuous behavior of the physical environment and this limits the expressiveness for the corresponding system. For another, AADL model is short of simulation and this is insufficient for AADL model to be analyzed.

The purpose of proposed AADL+ is to provide a comprehensive framework for the formal specification of cyber-physical systems both in architectural and behavioral aspects. The original AADL is responsible for architectural specification while the extended AADL+ focuses on behavioral description of cyber-physical systems. Furthermore, a plug-in to OSATE is developed to model and simulate cyber-physical systems. The advantage of AADL+ over AADL can be seen in Tab. 3.

Table 3 The advantage of AADL+ over AADL

	AADL+	AADL
Continuous behavior	√	×
Interactive behavior	√	×
Simulation	√	×
Expressiveness	Architecture + behavior	Only architecture

In the lunar rover control system, AADL is responsible for the architecture of the system through its core component. While AADL+ extends AADL with properties, variable declaration, equation component and interactive component. It is used to model the continuous behavior and interaction of the landing process. Furthermore, the lunar rover control system modeled with AADL+ can be simulated by linking with modelica.

8 Related Work

Lee and Seshia present detailed introduction for each components of CPS [1]. Modeling CPS based on SysML is proposed by academic institute and industry. Hause presents that SysML is a visual system modeling language that extends UML2 profile and accommodates the complex systems requirement specification, analysis, design, verification and validation [4]. Zhou Y, Baras J, Wang S [20] demonstrate a method to link model based system design with real-time simulations and analysis of the architecture model. They model Adaptive Cruise Control (ACC) System and refine the model. And then they transformed from the SysML model to an equivalent AADL model using model transformation. Furthermore, Cheddar is chosen as the external timing and scheduling analysis tool to analyze the XML model generated by the AADL model. IBM Corporation [21] has simulated cyber-physical systems using SysML and numerical simulation tools. They extend SysML to Simulink and make simulation in Simulink. A SysML based approach for modeling cyber-physical systems is proposed by Behjati R, Yue T, Nejati S, et al. [22] propose the ExSAM profile that extends SysML by adding AADL concepts to it. Using ExSAM and any SysML modeling environment is able to both model system engineering concepts and use AADL analysis tools. They implement ExSAM and evaluate its completeness and usefulness through some case studies. Bernardeschi C, et.al [23]describe a framework that facilitates modeling and simulation of cyber-physical systems. Software functionalities are modeled in the PVS theorem proving system. The characteristics of the plant are modeled in Simulink and full formal analysis of the software based on assume-guarantee reasoning. However, it is verbose to model the architecture of complex cyber-physical systems. Prist M, et.al [24] proposes a new simulation module to control a wireless cyber-physical system, by integrating LabVIEW and COOJA. But, It is only designed for the systems with wireless sensor and inadequate to simulate complex systems. NASA [25] proposes a model based system engineering to NASA's Space Communication Networks. Then it builds up a suit of methodology based on previous MBSE work done by INA architecture team. Several tools are evaluated for applying MBSE to SCaN, and many other programs and projects at NASA. Among them, SysML is chosen to model these projects. Tab. 4 shows the comparison of these approaches for modeling CPS.

Charon is a language concentrating on the modeling hybrid systems by means of the notions of agent and mode [26]. Individual components are described as agents in Charon Annex. Charon Annex provides agents with the operations of instantiation, hiding and parallel composition. Individual behaviors of agents (including the continuous dynamics and discrete controls) are described using modes. Charon Annex provides the operations of instantiation and nesting of modes in order to hierarchically describe the behaviors of agent. The key features of Charon Annex are as follows: 1) Architectural hierarchy: An agent is used to describe the system architecture, which is able to communicate with the environment by means of channels and shared variables; 2) Behavioral hierarchy: As a hierarchical state machine, a mode is used to describe the control flow that happens inside an agent. 3) Updates of continuous and discrete variables: The continuous variables in Charon Annex update their values continuously during the passage of time. The updates of discrete variables are specified by the transitions which connect the modes and labeled by guarded actions. The approaches adopted by AADL+ and Charon Annex for modeling the basic hybrid system are different, which is showed in Tab. 1. Alur, Grosu, et. al. propose the modular specification creativity in order to describe and model hybrid systems in Charon [18]. But, compared with AADL+ that we propose, Charon Annex has some disadvantages. Charon Annex only focuses on the modeling for hybrid systems, without containing the feature of modeling the interaction between cyber components and physical processes as AADL+ does. However, there is still no plug-in to OSATE supporting for Charon Annex yet, which limits its use.

Some methods are proposed to model cyber-physical systems with physical component and interactive behavior. An architecture-centric software development for cyber-physical systems has been proposed in [27] to describe CPS. They present the development process, and divide the process into four stages: modeling phrase, transforming and analysis phase, generation phase and integration phase. Then they use ROSLab for architecture modeling and transform the architecture to control system design. Core AADL language has been extended with a mechanism for discrete modeling and

Table 4 Approaches for modeling CPS

Tools	Idea	Continuous behavior	Model Cyber-Physical Interaction	Abstract level	Non-functional analysis
AADL+	Simulation& Verification	Yes	Yes	High	Yes
UML/MARTE	Multi-view	No	No	High	No
Simulink	Simulation	Yes	No	Low	No
SysML	Multi-view	No	No	High	No

analysis of control systems, but not for the continuous behavior of the physical environment. As a lightweight language extension to AADL, hybrid Annex [28] is introduced for continuous time modeling, fulfilling the need for integrated modeling of the computing system along with its physical environment. The Isolette system is used to illustrate continuous behavior modeling with the proposed Hybrid Annex. Bujorianu, et.al propose a model reasoning the problems to uncertainty of the physical environment to express the interaction between the cyber world and physical situation. Then they transfer the problem to model the automatic controllers. Eventually the context of CPS operating appears in mixed human/autonomous control [29]. Zhang, Dong, et. al. [30] propose a hierarchical and compositional modeling approach based on AADL and timed automata. They extend AADL by annex and separate the component abstraction of CPS model into three categories. Then they translate the CPS model into the networks of timed automata and use UPPAAL to analyze system model. Although they propose these transformation rules, it is short of the tool corresponding to these transformation rules. R. Alur maintains that the model for the embedded control systems is hybrid systems and the problem can be translated to a solution that combines the traditional state-machine with classical differential-equations, which is respectively based on models for discrete control and models for continuously evolving physical activities [31]. Sun and Zhou describe physical processes and their interactions through presenting an extension of AADL component, but it is short of providing its syntax and semantic [32]. Stefan D, et al. present the MECHATRONICUML tool suite to support the modeling and analysis of cyber-physical systems from formal requirement to platform independence software model [33]. Zhang extends standard AADL to incorporate new features such as spatial aspect, continuous dynamic and physical environment by the differential dynamic logic in railway cyber-physical systems [34]. However, the work can't guarantee the consistency of multi-aspect effectively. Ehsan A, Dong Y W, Brian L, et al. [35] talk about cyber-physical systems from the perspective of composition. They model the structure of the system using the core AADL constructs. The discrete behavior of the control system is modeled and verified using the BLESS annex. The continuous behavior of the system and the cyber-physical interaction are modeled using the Hybrid annex. The system level behavior is verified using the HHL Prover. Behavior constraints are specified using first-order logic formulas as Assertions. After modeling every part of the cyber-physical systems, they use theorem prover to ensure the correctness of the model. But it is insufficient to verify non-functional property of the system. Table 5 shows the comparison of these approaches for modeling CPS based on AADL.

Behavior Annex concentrates on the behavioral modeling of AADL like our AADL+. Yang, Hu, et. al. propose a formal semantics for the AADL behavior annex. The goal of this proposal is to enable the expression of high level composition notions, such as HRT-HOOD (Hard Real-Time Hierarchical Object Oriented Design), in AADL, extended by behavioral annexes using TASM (Timed Abstract State Machine). The behaviors described in Behavior Annex are based on state variables whose evolution is specified by transitions. However, Behavior Annex only can describe the discrete behavior in the systems, which is insufficient to model CPS [36]. Because of this insufficiency of Behavior Annex for modeling CPS, we decide to propose AADL+ to support not only discrete and continuous dynamics modeling, but also cyber-physical interaction modeling. Kamandi and Habibi review the history of developments on semantics frameworks for modeling languages [37]. Rajhans, Cheng, et. al. extend software architecture concepts to model physical environment and the interaction systems between cyber world and physical environment [14]. However, AADL+ is able to handle all of the above things.

Table 5 Comparison of Modeling CPS based on AADL

Literature	Methodology	Correctness	Contribution	Evaluation
AADL+	Simulation-based	Simulation	Continuous dynamics + Cyber-physical interaction + Differential equation + AADL core	Functional analysis + non-functional analysis
[20]	Component-based	Schedulibility		CanâĂŹt model continuous behavior
[22]	Component-based		Extending SysML with AADL concepts	Only specification
[30]	State-based	Model checking	AADL + Timed Automata + Differential equation	Short of the tool corresponding to these transformation rules
[34]	Aspect-oriented	Simulation and model checking	Spatial + Continuous dynamics + Dynamic logic + Timed CSP	CanâĂŹt guarantee the consistency of multi-aspect
[35]	Logical deduction	Theorem proving	BLESS + Hybrid Annex + HHL	Insufficient to verify non-functional property
[36]	State-based	Model checking	Mapping AADL model to TASM	

9 Conclusions and Future work

In this paper, we propose an AADL sublanguage called AADL+ for modeling cyber-physical systems. We provide the syntax and semantics of AADL+ and develop a plug-in to OSATE. The tool has the features of syntax checking and simulation for cyber-physical systems. To illustrate the proposed model language, we model the landing process of lunar rover and simulate its behavior through a link with Modelica.

As we mentioned before, AADL model is presented in three forms: AADL text, AADl XML and graphical AADL editor in OSATE. But the AADL+ plug-in to OSATE we developed only supports AADL text right now. We are planning to improve the plug-in to enable user to build AADL+ models not only in the form of AADL text, but also graphical AADL editor.

Moreover, we would like to extend our approach to schedulibility analysis affected by an increasing number of components. It would be interesting to check the effect of time in physical environment and cyber system.

Besides, we introduce assert in our work. Alloy [38] is a lightweight modelling language for software design and it includes assert and fact. It is amenable to a fully automatic analysis. It would be interesting to extend our approach to Alloy for checking the satisfiability.

While this work is preliminary, in order to study more systems with dynamic continuous property, we are planning to employ our approach to model the train control system, aero-engine control system and automatic driving system. We believe our early results show promise, and our work can provide a fundamental support to the modeling of cyber-physical systems.

Acknowledgements

This paper is partially supported by the projects funded by the National Key R&D Program of China 2017YFB1001801, NSFC Key Project 61332008, SHEITC160306 and NSFC 61572195.

Appendixes

A Mapping rules

AADL+ is able to simulate the model of cyber-physical systems, while AADL can't. We define the mapping rules to translate AADL+ model to modelica model. It will generate an intermediate file and the intermediate file is linked to Modelica simulator. We simulate the model through Modelica simulator. Fig. 14 is the corresponding rule:

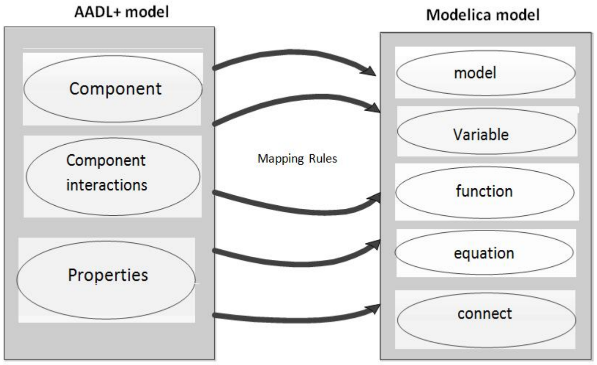


Fig. 14 Mapping rules diagram.

A.1 Data Component Mapping

In AADL+, we can use the declaration of data type, data implementation and data instance to model the data type in modelica. Data type can define the data associated with port. Data implementation defines the internal structure of the data type, and data instance defines the reference to data component by others. In modelica, data type is divided into two kinds: basic data type and defined data type through the key word type by developer. So, we capsulate the data component into two packages: basic_data_type and developer_data_type, which makes the hierarchy of the data clarified.

A.2 Equation and Function transformation

Modelica is different from other modelling or programming language because of its uniqueness of the operation based on equation. These equations beginning with key word equation define a series of operation and restrain to the internal attributes and variables. Function defines a set of sequential operation and algorithm. In AADL, subprogram component can represent source text or code snippet that executes sequentially. The component provides the operation to attribute and data operation or the corresponding service for these who calls the component. Because of the similarity of equation and function in syntax and properties, AADL+ can present several subprograms to every physical component.

However, a component is essential to call subprogram to execute the operation defined by subprogram. The thread component represents a schedulable unit that is executing sequentially and concurrency. However, the defined subprogram is an executable concurrent unit. Therefore, we distribute a thread for every physical meta component to the corresponding subprogram. The mapping rules can be seen in Table 6:

Table 6 Mapping rules diagram

AADL+ components	Modelica elements
system component	variable assignments
equations	equations
subprogram component	functions
thread component	call

A.3 Connections mapping

Modelica defines many interactive operations in subcomponent and between components including shared data, messages and signals. Also, AADL+ provides many interactions between components like port connection, data access and subprogram calls. So, it's necessary to present some rules for the interaction components.

A.3.1 Port Connection

A port represents a connection between two components and provides a communication for data or event. Modelica uses key word *connect* to concatenate the port by input and output declaration. The internal analyzer can generate a series of equation clause using the corresponding physical theorems. The rules are shown in Table 7:

Table 7 Port mapping rules

AADL+ components	Modelica elements
in data port	input
out data port	output

Table 8 The mapping rules of subprogram calls

AADL+ components	Modelica elements
thread implementation thread_name **calls**{ call_identifier: **subprogram** subprogram_name }; **end** thread;	execution of equation/function

A.3.2 Subprogram Calls

The interaction between components in Modelica mainly uses equation and function. In AADL+, the physical component contains equations to execute complex operations, while the cyber component doesn't. The subprogram component can calls other high order programming language code snippet to support the complicated operation in equations. The subprogram can be called by thread in AADL+ specification, so we map equation and function in modelica to subprogram in cyber component. These rules are shown in Table 8:

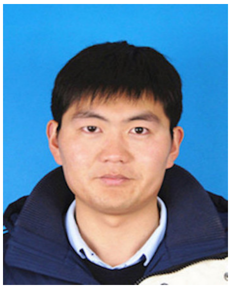
References

1. Lee E A, Seshia S A, Introduction to embedded systems: A cyber-physical systems approach. Lee & Seshia, 2011.
2. Debbabi M, Hassaïne F, Jarraya Y, Soeanu A, Alawneh L. Unified modeling language. Encyclopedia of Systems Biology. 2010, 20(1):9.
3. Gerard S Selic B. Modeling and analysis of real-time and embedded systems with uml and marte : developing cyber-physical systems. Elsevier, 2014.
4. Cao Y, Liu Y S, Paredis C J J. System-level model integration of design and simulation for mechatronic systems based on SysML. Mechatronics, 2011, 21(6):1063-1075.
5. Feiler P H, Lewis B A, Vestal S. The SAE architecture analysis & design language (AADL) a standard for engineering performance critical systems. In: Proceedings of Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control. IEEE Xplore, 2006,1206-1211.
6. Vestal S. MetaH support for real-time multi-processor avionics. In: Proceedings of Joint Workshop on Parallel and Distributed Real-Time Systems. IEEE Computer Society, 1997:11.
7. Lee E A. The past, present and future of cyber-physicalsystems: a focus on models. Sensors, 2015,15(3):4837-4869.
8. Wang Z J, Xie L L. Cyber-physical systems: a survey. Acta Automatica Sinica, 2011, 37(10): 1157-1166.
9. Feiler P H, Gluch D P. Model-based Eegineering with AADL: an introduction to the SAE architecture analysis & design language. Addison-Wesley Professional, 2012.
10. SAE Aerospace. Sae as5506 annex: Behavior specification v1.6.
11. SAE Aerospace. Sae as5506a: Architecture analysis and design language v2.0.
12. Oksendal B. Stochastic differential equations - an introduction with applications. Springer-Verlag, 2006.
13. Modelica Association. Modelica- a united object-oriented language for systems modeling-language specification version 3.3. 2014.
14. Rajhans A, Cheng S W, Schmerl B, David Garlan, Krogh B H, Agbi C, Bhave A. An architectural approach to the design and analysis of cyber-physical systems. 2009, 21:14-38.
15. Banerjee A, Kandula S, Mukherjee T, Gupta S. BAND-AiDe: A tool for cyber-physical oriented analysis and design of body area networks and devices. Acm Transactions on Embedded Computing Systems. 2012, 11(S2):49.
16. Smith D B, Hanlen L W. Channel modeling for wireless body area networks. Ultra-Low-Power Short-Range Radios. Springer International Publishing, 2015,25-55.
17. Singhoff F, Legrand J, Nana L. Cheddar:a flexible real time scheduling framework. Acm Sigada Ada Letters. 2004, XXIV(4):1-8.
18. Alur R, Grosu R, Hur Y, Kumar V, Lee I. Modular specification of hybrid systems in Charon. Hybrid Systems: Computation and Control. Springer Berlin Heidelberg, 2000,6-19.
19. Sokolsky O. The montana toolset: Osate plugins for analysis and code generation. In: Proceedings of AADL Workshop, 2005.
20. Zhou Y C, Baras J, Wang S G. Hardware software co-design for automotive cps using architecture analysis and design language. 2016.
21. Palachi E, Cohen C, Takashi S. Simulation of cyber physical models using SysML and numerical solvers. In: Proceedings of Systems Conference. 2013,671-675.
22. Behjati R, Yue T, Nejati S, Briand L, Selic B. Extending SysML with AADL concepts for comprehensive system architecture modeling. In: Proceedings of European Conference on Modelling Foundations and Applications. 2011,236-252.
23. Bernardeschi C, Domenici A, Masci P. A PVS-Simulink integrated environment for model-based analysis of cyber-physical systems. In: Proceedings of IEEE Transactions on Software Engineering, 2017, PP(99):1-1.
24. Prist M, Freddi A, Longhi S, Monteriù A. An integrated simulation module for wireless cyber-physical system. In: Proceedings of International Conference on Environment and Electrical Engineering. 2015,1397-1402.
25. Bhasin K, Barnes P, Reinert J, Golden B. Applying model based systems engineering to NASA's space communications networks. In: Proceedings of Systems Conference. 2013,325-330.
26. Lee J, Cha R, Han Y H, Nam W, Choi J Y, Kim W T, Park S M. Modeling autonomous military robots using hybrid system framework. In: Proceedings of International Conference on Information and Communication Technology Convergence. 2010:429-430.
27. Sokolsky O, Pajic M, Bezzo N, Lee I. Architecture-centric software development for cyber-physical systems. In: Proceedings of the First Workshop on Cyber-Physical System Architectures and Design Methodologies (CPSArch 2014), 2014.
28. Ahmad E, Larson B R, Barrett S C, Zhan N J, Dong Y W. Hybrid annex: an aadl extension for continuous behavior and cyber-physical interaction modeling. Acm Sigada Ada Letters. 2014, 34(3):29-38.
29. Bujorianu M C, Bujorianu M L, Barringer H. A unifying specification logic for cyber-physical systems. In: Proceedings of Mediterranean Conference on Control and Automation. 2009,1166-1171.
30. Zhang Y, Dong Y W, Zhang F, Zhang Y F. Research on modeling and analysis of CPS. In: Proceedings of International Conference on Autonomic and Trusted Computing. 2011,92-105.
31. Alur R. Formal verification of hybrid systems. In: Proceedings of the International Conference on Embedded Software. 2011,273-278.
32. Sun Z H, Zhou X S. Extending and Recompiling AADL for CPS Modeling. In: Proceedings of IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing. IEEE Computer Society, 2013,1225-1230.
33. Dziwok S, Gerking C, Becker S, Thiele S, Heinzemann C, Pohlmann U. A tool suite for the model-driven software engineering of cyber-physical systems. In: Proceedings of ACM Sigsoft International Sym-

posium on Foundations of Software Engineering. 2014,715-718.

34. Zhang L C. Aspect-oriented modeling of railway cyber physical systems based on the extension of aadl. In: Proceedings of IEEE International Conference on High Performance Computing and Communications (HPCC) & IEEE International Conference on Embedded and Ubiquitous Computing (EUC), 2013:2104-2111.
35. Ehsan A, Dong Y W, Brian L, Tang T, Lü J D, Zhan N J. Behavior modeling and verification of movement authority scenario of Chinese Train Control System using AADL. Science China Information Sciences, 2015, 58(11):112104-112104.
36. Yang Z B, Hu K, Ma D F, Pi L. Towards a formal semantics for the AADL behavior annex. 2009:1166-1171.
37. Kamandi A, Habibi J. A survey of syntax and semantics frameworks of modeling languages. In: Proceedings of International Conference on Computer Science and ITS Applications. 2009:1-6.
38. Torlak E, Taghdiri M, Dennis G, Near J P. Applications and extensions of Alloy: past, present and future. Mathematical Structures in Computer Science, 2013, 23(4):915-933.

Jing Liu is currently a professor of computer science at East China Normal University (ECNU), China. She is also the vice director of Computing Theories Institute, ECNU. In recent years, she is working on the area of model driven architecture. Now her work focuses on the design of real-time embedded systems and cyber physical system.



Tengfei Li received the B.S. degree from the School of Mathematics and Statistics, Changshu Institute of Technology, Suzhou, China, in 2014. He is currently a Ph.D. student in the School of Computer Science and Software Engineering, East China Normal University, Shanghai, China. His research interests are in the area of safety-critical cyber physical systems, formal verification and hybrid systems.

Zuohua Ding received his PhD (1996) in Mathematics(advisor: Professor Athanassios G. Kartsatos) and MS (1998) of Computer Science (advisor: Professor Abraham Kandel) all from University of South Florida, USA. He is currently a professor and the director of Lab of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, China, and he has been a research professor of National Institute for Systems Test and Productivity, USA since 2001. From 1998 to 2001, he was a senior software engineer in Advanced Fiber Communication, USA. His research interests include system modeling, program analysis, service computing, software reliability prediction and Petri nets, subjects on which he has authored and coauthored more than 70 papers.

Yuqing Qian received the B.E. degree and M.E. degree from School of Computer Science and Software Engineering, East China Normal University, Shanghai, China, in 2011 and 2014, respectively. He is currently an engineer in internet finance. His research interests are in the area of program analysis and software engineering.

Haiying Sun was born in 1976. She received the master's degree in computer science and technology from the NanJing University Science and Technology in 2001. She earned her PhD in Formal Method From East China Normal University in 2017. She is now a lecturer at East China Normal University. Her research interests include formal method, automatic test generation, system simulation and model-driven engineering.

Jifeng He is currently a professor of computer science at East China Normal University (ECNU), China. He is an academician of Chinese Academy of Sciences. He is also the dean of Software Engineering Institute, ECNU. In recent years, he has also been working on the mathematical model about the co-design of software and hardware. His work focuses on design of real-time embedded systems, cyber physical system, and the Internet of Things.