Quantitative Performance Evaluation of Uncertainty-Aware Hybrid AADL Designs Using Statistical Model Checking

Yongxiang Bao, Mingsong Chen* *Member, IEEE*, Qi Zhu *Member, IEEE*, Tongquan Wei *Member, IEEE*, Frederic Mallet *Member, IEEE* and Tingliang Zhou

Abstract—**Hybrid Architecture Analysis and Design Language (AADL) has been proposed to model the interactions between embedded control systems and continuous physical environment. However, the worst-case performance analysis of Hybrid AADL designs often leads to overly pessimistic estimations, and is not suitable for accurate reasoning about overall system performance, in particular when the system closely interacts with an uncertain external environment. To address this challenge, this paper proposes a statistical model checking based framework that can perform quantitative evaluation of uncertainty-aware Hybrid AADL designs against various performance queries. Our approach extends Hybrid AADL to support the modeling of environment uncertainties. Furthermore, we propose a set of transformation rules that can automatically translate AADL designs together with designers' requirements into Networks of Priced Timed Automata (NPTA) and performance queries, respectively. Comprehensive experimental results on the Movement Authority (MA) scenario of Chinese Train Control System Level 3 (CTCS-3) demonstrate the effectiveness of our approach.**

Index Terms—**Hybrid AADL, Uncertainty, Statistical model checking, Quantitative performance evaluation.**

I. Introduction

To promptly and accurately sense and control the physical world, more and more real-time embedded systems are deployed into our surrounding environment. As a result, the stringent safety-critical requirements coupled with increasing interactions with uncertain physical environments make the design complexity of *Cyber-Physical Systems* (CPS) skyrocketing [1], [2]. Unfortunately, due to the lack of architecture-level performance evaluation approaches considering uncertain environments, the required performance of integrated CPS implementations can be easily violated. Therefore, how to model the uncertain behaviors of both cyber and physical elements and how to guarantee meeting the critical functional and real-time requirements have become major challenges in CPS architecture design.

Architecture Analysis and Design Language (AADL) [3], [4], [5] has been widely adopted for the design and analysis of safety-critical real-time systems (e.g., automotive, avionics and railway systems). By defining various modeling constructs for hardware and software components, AADL core language supports the structural description of system partitioning and connectivity among components, while the semantics of AADL can be extended via annex sublanguages and user-defined properties. An AADL specification provides a set of modeling constructs for the description and verification of both functional and non-functional properties of interacting software and hardware components. Since the core AADL language only supports modeling of hardware and software components, to model the physical environment we adopt the Hybrid AADL, which supports continuous behavior modeling via the Hybrid annex [6].

When modeling a safety-critical system using AADL, before the design refinement, there is a rigorous certification process to verify whether the AADL design satisfies the required safety properties. Although existing AADL IDE tools such as OSATE [7] can be used to check timing properties (e.g., flow latency), most of existing approaches adopt the worst-case timing analysis without considering performance variations, which can easily lead to overly pessimistic performance estimations. To extend the performance analysis capability of AADL designs, various model transformation approaches [8], [9] have been proposed to verify AADL models based on existing verification and analysis tools. For quantitative analysis of AADL designs with uncertain environment, designers would like to ask questions such as *"What is the probability that a specified scenario can be achieved within time x?"*. However, existing approaches focus on checking safety properties that only have an answer of "yes" or "no" without considering uncertain environments. Few of them can quantitatively reason why a given performance requirement cannot be achieved and answer how to improve the design performance. *Clearly, the bottleneck is the lack of powerful quantitative evaluation approaches that can help AADL designers to make decisions during the architecture design.*

To enable the quantitative analysis for Hybrid AADL designs, we propose a novel framework based on *Statistical Model Checking* (SMC) [10] which relies on the monitoring of random simulation runs of systems. By analyzing the simulation results using statistical approaches (e.g., sequential hypothesis testing or Monte Carlo simulation), the satisfaction probability of specified properties (i.e., performance requirements) can be estimated. Unlike traditional formal verification methods, which need to explore all the state space, SMC only investigates a limited number of simulation runs of systems and requires far less memory and time. Therefore, SMC is

The authors Yongxiang Bao, Mingsong Chen, and Tongquan Wei are with Shanghai Key Laboratory of Trustworthy Computing at East China Normal University, Shanghai, 200062, China. The author Qi Zhu is with Department of Electrical and Computer Engineering at University of California, Riverside, CA 92521, USA. The author Frederic Mallet is with Université Nice Sophia Antipolis, F-06902 Sophia Antipolis Cedex, France. Tingliang Zhou is with Casco Signal Ltd., Shanghai, 200070, China.

This work was partially supported by the grants from Natural Science Foundation of China (Nos. 61672230 and 91418203), Shanghai Municipal NSF 16ZR1409000, and National Science Foundation of United States (CCF-1553757 and CCF-1646381).

*Corresponding author. Tel: +(8621) 62235116; fax: +(8621) 62235255; E-mail: mschen@sei.ecnu.edu.cn.

very suitable for the approximate functional validation of complex AADL designs. We use the statistical model checker UPPAAL-SMC [10] as the engine of our approach, to leverage its rich modeling constructs and flexible mapping mechanisms.

Based on UPPAAL-SMC, this paper makes **three following major contributions**: i) We extend the syntax and semantics of Hybrid AADL specifications [6] using our proposed *Uncertainty annex*, which enables the accurate modeling of both performance variations caused by uncertain environments and performance requirements specified by designers. ii) To automate the quantitative analysis of uncertainty-aware Hybrid AADL designs, we rely on *Network of Priced Timed Automata* (NPTA) [10] as the model of computation in our approach. We propose a set of mapping rules that can automatically transform uncertain-aware Hybrid AADL designs into NPTA models and convert the performance requirements into various kinds of queries in the form of cost-constrained temporal logic [11]. iii) Based on our proposed SMC-based evaluation framework, we implement a tool chain that integrates both UPPAAL-SMC and the open-source AADL tool environment OSATE to enable the automated performance evaluation and comparison of uncertainty-aware Hybrid AADL designs.

The rest of this paper is organized as follows. After introducing the related work on AADL and SMC-based evaluation approaches in Section II, Section II presents the details of our proposal. Based on an industrial CTCS-3 MA design, Section IV shows that our proposed approach can be effectively applied to the quantitative analysis of Uncertain Hybrid AADL designs. Finally, Section V concludes the paper.

II. Related Work

To facilitate architecture design and analysis of safety-critical systems, various AADL simulation and verification tools were investigated [5]. For example, Jahier et al. [15] proposed an approach that can translate both AADL models and software components developed in synchronous languages (i.e., SCADE, Lustre) into executable models, which can be simulated and validated together. In [13], Yu et al. presented a co-simulation and co-verification framework for AADL and Simulink designs. Based on a formal polychronous/multi-clock model of computation, an original clock-based timing analysis and validation of the overall system is achieved. In [16], Larson et el. introduced the Behavioral Language for Embedded Systems with Software (BLESS) annex for AADL. The extended AADL language enables engineers to specify contracts on AADL components that can capture both functional and timing properties. They also developed the BLESS proof tool which can check whether AADL behavioral descriptions conform to specified contracts. Although these approaches are promising in functional checking, few of them consider performance issues for AADL designs.

Rather than developing dedicated verification tools for AADL designs, more and more model transformation-based AADL analysis approaches resort to the benefits of widely-used model checking techniques [18]. For instance, Hu et al. [8] presented a set of formally defined rules that can translate a subset of AADL to corresponding *Timed Abstract State Machines* (TASM) models for the purpose of timing and resource verification. To ensure completeness and consistency of an AADL specification as well as its conformity with the end product, Johnsen et al. [9] presented a formal verification technique by translating AADL designs to timed automata models. In [12], Bozzano et al. proposed a formal semantics for AADL that incorporates functional, probabilistic and hybrid aspects. Based on model checking techniques, they developed a toolset that can be used for a wide spectrum of design purposes ranging from requirements validation to performability evaluation and diagnosability analysis. Although the above model checking-based methods can check the functional correctness of systems in a fully automated manner, most of them suffer from the state space explosion problem [18]. Moreover, very few of them take the uncertain physical environment into account.

Since cyber-physical systems interact with surrounding physical environment frequently, the behavior modeling and verification of multirate and hybrid systems have become important research topics in AADL design. For example, Bae et al. [14] proposed a modeling language named Multirate Synchronous AADL, which can be used to specify multirate synchronous designs using existing AADL modeling standard. They also defined the formal semantics of Multirate Synchronous AADL, which enables the formal verification using Real-Time Maude. Based on BLESS annex [16] and Hybrid annex [6], Ahmad et al. [20] modeled and analyzed the movement authority scenario of the Chinese Train Control System Level 3 (CTCS-3) in AADL. Their approach can verify both discrete and hybrid behaviors of annotated Hybrid AADL designs based on the interactive *Hybrid Hoare Logic* theorem prover [17]. However, since their approach is based on theorem proving methods, it cannot be fully automated due to the required expert knowledge and manual "proof assistants". Furthermore, existing theorem proving based methods focus on proving functional correctness of AADL designs. Few of them can be used to evaluate design performance within uncertain physical environment.

Due to its scalability and effectiveness in evaluating stochastic behavior, statistical model checking has become a preferred option in performance analysis of system designs with uncertainties [21]. Since statistical model checking is based on simulation, it requires far less memory and time, which enables highly scalable validation for AADL designs. For example, Bruintjes et al. [24] introduced a statistical model checking approach for timed reachability analysis of extended AADL designs. They developed a simulator that can perform probabilistic analysis of underlying stochastic models using Monte Carlo simulation. Our approach differs greatly from [24]. In [24], the extended AADL is based on linear-hybrid models, whereas our approach supports the modeling of nonlinear behaviors for a large group of CPSs. In particular, the clock rates in UPPAAL-SMC can be described using ordinary differential equations [10], e.g., $c1' == sin(c2)$, $c1' == c1 * c2 + c3$, where $c1$, $c2$ and $c3$ are three clock variables. In addition to the capability of modeling nonlinear behaviors, our approach focuses on evaluating CPS performance under uncertain environments, while [24] emphasizes on the error behavior modeling of hardware

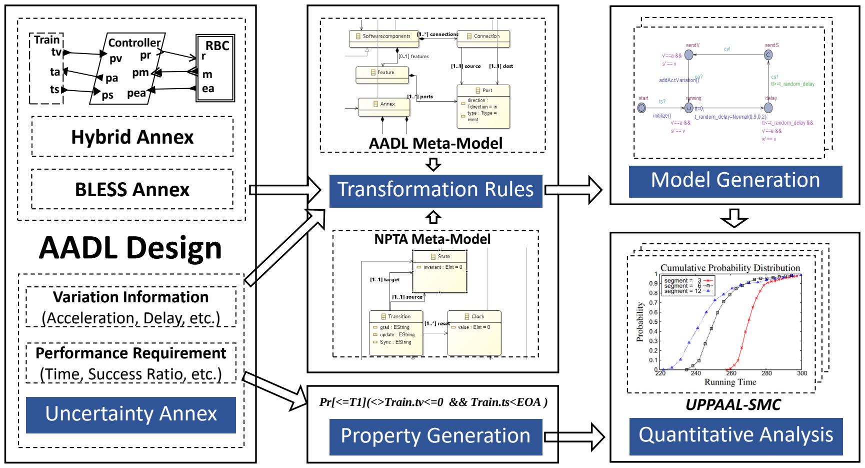


Fig. 1. Workflow of our framework

and software components. Moreover, [24] only considers the probability of event occurrences and delay variations following either uniform or exponential distributions, while our approach allows designers to define their own uncertain objects (e.g., system parameters, user inputs) following a wide spectrum of programmable distributions. Furthermore, the method in [24] only supports the evaluation of time-bounded queries, while our evaluation approach is based on cost-constrained temporal logic which is more comprehensive.

To the best of our knowledge, so far there is no approach that supports the performance evaluation for Hybrid AADL designs considering the uncertainties caused by physical environments. Our proposed approach is the first attempt that not only supports the uncertainty modeling in AADL, but also enables the quantitative performance reasoning and comparison of uncertainty-aware designs at the architecture level.

III. Our Approach

Figure 1 shows the workflow of our approach. Since the core AADL focuses on structural modeling, to describe concrete execution behaviors of components, we need to resort to annex modeling which is a mechanism provided by AADL for the purpose of semantics extension. In this paper, we focus on uncertain hybrid systems, thus our approach adopts the Hybrid and BLESS annexes to describe the dynamic and hybrid behaviors of systems. To extend the semantics of hybrid systems, we propose the *Uncertainty annex* to specify various performance variations (e.g., network delays, sensor inputs) and performance requirements posed by designers. Based on our defined AADL and NPTA meta-models, Hybrid AADL designs with extended performance variation information can be extracted and transformed into corresponding uncertainty-aware NPTA models. The specified performance requirements are also parsed by our developed parser for the generation of properties, which are in the form of cost-constrained temporal logic [10]. Based on statistical model checker UPPAAL-SMC, our approach can conduct the quantitative evaluation of Uncertain Hybrid AADL designs against various properties (i.e., performance and safety queries). In the following subsections, we explain the major steps of our approach in details.

A. Uncertainty-Aware Modeling of Hybrid AADL

To model a hierarchical real-time system, a typical AADL [3], [4] design comprises both software components and their corresponding execution platform. Software components such as *thread*, *thread group*, *process*, *data* and *subprogram* can be used to construct the software architecture of systems. Execution platform components including *processor*, *memory*, *device* and *bus* can be used for hardware modeling. Within a system, all these components communicate with each other through *connections* to accomplish specific functions.

Definition 3.1: An *Uncertain Hybrid AADL* design is a 9-tuple $<Comp, P_{ort}, C_{onn}, M_p, D, \Sigma, M_\Sigma, A_{nnex}, M_a>$ where: i) *Comp* is a finite set of hardware/software components including their *declarations* and *implementations*; ii) P_{ort} is a finite set of component ports including *data ports*, *event ports* and *event data ports*; iii) $C_{onn} \subseteq P_{ort} \times P_{ort}$ denotes a finite set of *connections* between ports; iv) $M_p : P_{ort} \to Comp$ assigns ports to corresponding components; v) D is a finite set of data which can be transferred via connections; vi) Σ is a finite set of *AADL properties*; vii) $M_\Sigma : \Sigma \to Comp$ assigns *AADL properties* to corresponding components; viii) A_{nnex} is a finite set of annex declarations, i.e., *BLESS annex*, *Hybrid annex*, and *Uncertainty annex*; ix) $M_a : A_{nnex} \to Comp$ maps annexes to their components. ■

To enable the quantitative evaluation of Hybrid AADL designs considering uncertain environments, Definition 3.1 gives the formal definition of our *Uncertain Hybrid AADL*. In AADL, the definitions of both hardware and software components contain two parts, i.e., *declaration* and *implementation*. To enable interactions with other components, *declaration* defines *ports* for components which can be used to transmit and receive data or events, whereas *implementation* provides the details of a component including its subcomponents, properties and the connections between ports. In addition to basic data types, AADL allows designers to define their own data types to enrich AADL designs.

By using annexes, designers can precisely define and interpret behaviors of components by themselves. Different from traditional AADL designs, our Uncertain Hybrid AADL is based on a combination of BLESS, Hybrid and Uncertainty annex declarations. Our approach adopts BLESS annexes and Hybrid annexes to model the discrete and continuous behaviors of AADL components, respectively. To model various uncertainties caused by external environment, we introduce the Uncertainty annex.

1) Background of BLESS and Hybrid Annexes: Based on state machine like semantics, BLESS annex [16] provides a set of notations which can be used to formally define discrete component behaviors, while the BLESS assertions can be used to specify and check the desired system properties. Definition 3.2 gives the formal definition of a BLESS Annex Instance (BAI) which can be embedded into a component implementation. Note that a transition of a BAI may have multiple actions for variable assignments or port communications. Since our approach does not adopt the assertions provided by BLESS annex, we did not incorporate it in Definition 3.2.

Definition 3.2: A *BLESS Annex Instance* [16] is a 6-tuple$< S, s_0, BV, Act, G, T >$ where, i) S is a finite set of states; ii) s_0 is the initial state; iii) BV is a finite set of variables; iv) Act is a finite set of actions; v) G is a finite set of guard conditions over BV; and vi) $T \subseteq S \times G \times 2^{Act} \times S$ denotes the finite set of transitions. ■

Definition 3.3 gives the formal definition of Hybrid annex instances. Based on semantics of Hybrid CSP (Communicating Sequential Processes) [6], [27], Hybrid annex can be applied in continuous behavior modeling of AADL *device* and *abstract* component implementations, such as sensors, actuators and physical processes. When using the Hybrid annex, both discrete and continuous variables are declared in the *variables* section, and the values of constants are initialized in the *constants* section. The *behavior* section of a Hybrid Annex Instance (HAI) is used to describe the continuous behaviors of annotated AADL components in terms of concurrently-executing processes. Such behaviors indicating continuous process evolutions are specified using differential expressions. The physical processes communicate with each other using channels (declared in the *channels* section) or ports (i.e., the ports of associated AADL component). Continuous process evolution may be terminated after a specific time or on a communication event, which is 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 place along any one of the named channels or ports. For more details of the processes and interrupts, please check the examples shown in Listing 1 and the model transformation rules in Section III-B5. Note that the Hybrid annex also supports assertions which take the same format as BLESS assertions [16]. Since none of these assertions are suitable for quantitative analysis, we neglect the assertion definition in Definition 3.3.

Definition 3.3: A *Hybrid Annex Instance* is a 6-tuple $< HV, HC, P, CP, I, M_i >$ where, i) HV is a finite set of discrete and continuous variables; ii) HC is a finite set of constants that can only be initialized at declaration; iii) P is a finite set of processes that are used to specify continuous behaviors of AADL components; iv) CP is a finite set of channels and ports for synchronizing processes; v) I is a finite set of time or communication interrupts; and vi) $M_i : I \rightarrow P$ binds interrupts to associated processes. ■

2) Definition, Syntax and Semantics of Uncertainty Annex: Although there are many tools that are proposed to check the performance of AADL designs, most of them assume a uniform distribution for flow delays. Few of them consider the variations (e.g., sensor inputs, network delays) caused by uncertain environments. To support the modeling of such kinds of uncertainties, based on Definition 3.4, we extend the semantics and syntax of Hybrid AADL using our proposed Uncertainty annex.

Definition 3.4: An *Uncertainty Annex Instance* (UAI) is a 7-tuple $< TV, PV, DIST, M_{tv}, M_{pv}, M_{dist}, Q >$ where, i) TV is a finite set of stochastic *time* variables; ii) PV is a finite set of stochastic *price* variables; iii) $DIST$ is a finite set of distribution functions; iv) $M_{tv} : TV \rightarrow \{P_{ort} \cup T\}$ binds each time variable $tv \in TV$ to a port $p \in P_{ort}$ or a transition $t \in T$; v) $M_{pv} : PV \rightarrow \{BV \cup HC\}$ binds each variable $pv \in PV$ to a variable $bv \in BV$ or a constant $hc \in HC$; vi) $M_{dist} : \{TV \cup PV\} \rightarrow DIST$ assigns each variable $v \in \{TV \cup PV\}$ with a distribution function; and vii) Q is a finite set of queries for the quantitative performance evaluation. ■

Unlike existing approaches, our Uncertainty annex supports a large spectrum of distributions which can be used to accurately capture the system behaviors within an uncertain environment. To simplify the stochastic behavior modeling, we define two kinds of stochastic variables, i.e., *time variables* which denote the time variations of AADL constructs (e.g., ports), and *price variables* that indicate the value variations of AADL variables and constants. For example, if a time variable tv is bound to a port p, the data transmission time via p follows the specified time distribution $M_{dist}(tv)$. Note that Uncertainty annex itself is only a syntactical notation, which specifies the uncertain values for constants and variables and uncertain communication delays for ports. It does not change the semantics of Hybrid AADL designs. To enable the statistical model checking, the Uncertainty annexes are used to indicate the value of constants/variables or the delay of ports at the beginning or during the execution of a simulation run following the give distributions. During the automated quantitative evaluation, Uncertainty annex allows the designers to specify their queries to assess whether an Uncertain Hybrid AADL design satisfies the requirements.

$$
\begin{array}{l}
\mathit{Uncertainty\ Annex} ::= \{** \\
\quad \textbf{variables} \quad \{\mathit{variables\_declaration}\}+ \\
\quad \textbf{distributions} \quad \{\mathit{distribution\_declaration}\}+ \\
\quad \textbf{queries} \quad \{\mathit{query\_declaration}\}+ \\
**\}
\end{array}
$$

As an extension, UAIs can be embedded into AADL components as a subclause to specify their uncertain behaviors. To describe the context-free syntax of UA, we explain all the notations of Uncertainty annex using the *Extended Backus-Naur Form*, where *literals* are printed in bold; *alternatives* are separated by "|"; *grouping* are enclosed with parentheses"( )";

square braces "[ ]" delimit *optional elements*; and "{ }+" and "{ }∗" are used to signify *one-or-more*, and *zero-or-more* of the enclosed elements, respectively. As shown in the above production rule, an Uncertainty annex consists of three parts, i.e., *variables* section, *distributions* section and *queries* section. Their functions and usages are explained as follows.

Variables section: Instead of modeling the uncertainties of environment components directly, our approach implicitly reflects the environment uncertainties by specifying distributions for both data transmission time via connections between interconnected ports and the values of system parameters (e.g., AADL variables and constants). To model stochastic behaviors of systems, we define local variables in this section to indicate the uncertain dynamics of the corresponding AADL component features. In our approach, all these variables are associated with specific probability distributions to signify their possible values within an uncertain environment. The following rules show the grammar for *variables* section.

$$\begin{aligned} &variables\_declaration \ ::= \\ &\quad type\_prefix \ \{variable\_identifier\}+ \\ &\qquad\qquad \textbf{applied to} \ \{component\_ref\}* \\ &\quad type\_prefix \ ::= \ \textbf{time} \mid \textbf{dynamic price} \mid \textbf{static price} \\ &\quad component\_ref \ ::= \ features\_ref \mid annex\_subclause \end{aligned}$$

In the above rules, the type prefix *time* means that local variables are used to model the stochastic timing information of component features. For example, a local time variable can be bound to a component port to specify uncertain communication delays on the *connection* via this port. The local variables with type prefix *static/dynamic price* can be used to specify the uncertain value assignment for variables and constants declared in annotated components or their annexes. In our approach, we consider two kinds of local price variables, i.e., static price variables and dynamic price variables. Here, *static price* means that the initial value of the associated AADL (or annex) variables and constants are assigned stochastically at the beginning of system execution. Unlike *static price variables* which only conduct the initialization of variables or constants once, local *dynamic price variables* are usually bound to AADL (or annex) variables to model their random value updates when newly referred.

Distributions section: The *distributions section* is to specify the probability distributions of the variables defined in the *variables section*. To allow the modeling of various stochastic behaviors, our Uncertainty annex has a built-in distribution functional library that supports a large spectrum of widely used distributions, such as uniform, exponential and normal distributions. The following production rules show how to bind a variable with a specific distribution function.

$$\begin{aligned} &distribution\_declaration ::= \\ &\quad varable\_identifier\_reference = distribution \\ &distribution ::= \textbf{Normal}`('const, const`)' \\ &\qquad \mid \textbf{Uniform}`('const, const`)' \\ &\qquad \mid \textbf{Exponential}`('const`)' \mid ... \end{aligned}$$

Queries section: To quantify the performance of Uncertain Hybrid AADL models during the architecture level design, designers would like to ask questions such as "what is the probability that a scenario can happen or a condition can be satisfied with limited resources?". Uncertainty annex provides the *queries section* that can be used for declaring such queries to enable safety and performance evaluation of AADL designs. As an effective way to check the quality and performance of AADL designs, the designers can put their design requirements in this section. Only when all the evaluated requirements meet design targets, the AADL design can be used as a reference for the implementation.

$$\begin{aligned} &query\_declaration ::= \\ &\quad query\_identifier = query\_target \ \textbf{under} \ constraint \\ &query\_target ::= expr \ \{\textbf{\&\&} \ expr\}* \\ &expr ::= condition \mid `(' \ condition \ ( \ \textbf{\&\&} \mid \| \ ) \ expr \ `)' \\ &condition ::= identifier \quad operation \quad (\ const \mid identifier \ ) \\ &constraint ::= [\ identifier \ ] \ operation \ const \\ &operation ::= < \mid \leq \mid \textbf{==} \mid \textbf{!=} \mid \geq \mid > \end{aligned}$$

When specifying a query, designers should provide two things: i) a query target that denotes a safety scenario or performance metric in the form of a predicate expression; and ii) a constraint indicating the available resources to achieve the target. The above production rules present how to declare queries. Here, *identifier* denotes the name of AADL features (e.g., ports) or annex variables declared in annotated component implementations, and *const* denotes a constant value. The target of a query is a predicate represented by a conjunction of *expressions*. The query constraint is in the form of "*res op lim*", where *res*, *op* and *lim* denote resource, operation and resource limit, respectively. If the resource is not specified explicitly, the system time will be used as the resource by default.

Listing 1 presents a Hybrid AADL design example annotated with an uncertainty annex instance, which describes the uncertain behaviors of the train component within the CTCS-3 Movement Authority scenario (see details in Section IV). To operate safely, the train periodically sends its current location and velocity information to the on-board train controller through ports *ts, tv* and receives the acceleration instruction directed by the controller from the port *ta*. All these ports are defined in the *Train* declaration. To model the continuous behaviors, the train design adopts the Hybrid annex. Within the Hybrid annex, the system time is modeled using the continuous variable t whose rate is 1 (indicated by notation '**DT** 1 t=1' which means the derivation of t is 1). In the Hybrid annex behavior section, the notation '**DT** 1 s=v' defines train speed and the notation '**DT** 1 v=$a+fr$' denotes the acceleration of the train. During the running, the traction control force is determined by the value of a calculated by the controller, while the resistance is determined by the friction coefficient of the track. In this example, we consider two kinds of uncertainties. The first one represents uncertain communication delays between trains and controllers. The second one is the track friction which is highly dependent on the external environment (e.g., weather, temperature). Therefore, we define two local variables in the variables section of the uncertainty annex. The time variable *v_delay* is bound to the port *ts* with a distribution *Normal(0.15,0.04)* (defined in the *distributions* section). We set the variable *v_fr* as a static price variable, since we assume that the coefficient for the whole track and its value is only updated once at the beginning of each system run. In the queries section, query *p*1 tries to figure out the probability that the train can stop before the end of authority (denoted by EOA) within 300 seconds, and query *p*2 tries to reason whether the train can run 4 kilometers within 200 seconds.

```
abstract Train
    features
    ts: out data port CTCS_Types::Position;
    tv: out data port CTCS_Types::Velocity;
    ta: in data port CTCS_Types::Acceleration;
end Train;

abstract implementation Train.impl
annex Uncertainty {**
    variables
    time v_delay applied to Train.ts
    --  modeling connection delay
    static price v_fr  applied to Train.fr
    -- modeling track friction
    distributions
    v_delay = Normal(0.15,0.04)
    v_fr   = Normal(-0.1,0.05)
    queries
    p1 = Train.v<=0 && Train.s<EOA
     &&  Train.s>0 under <=300
    p2 = Train.s >= 4000 under <=200
**};

annex hybrid {**
    variables
    s : CTCS_Types::Position -- train position
    v : CTCS_Types::Velocity -- train velocity
    a : CTCS_Types::Acceleration -- train acceleration
    t : CTCS_Types::Time   -- system time
    fr : CTCS_Types::Deceleration -- track friction
    behavior
    Train ::= 'DT 1 s=v' & 'DT 1 v=a+fr' & 'DT 1 t=1'
          [[> ts!(s), tv!(v),ta?(a)]]> Continue
    Continue ::= skip
    RunningTrain ::= s:=0 & v:=0 & a:=0 & REPEAT(Train)
**};
end Train.impl;
```

Listing 1. Uncertain Hybrid AADL Design for *Train* in CTCS-3 MA

B. NPTA Generation from Uncertain Hybrid AADL

To formalize the semantics of Uncertain Hybrid AADL, we adopt NPTA as the model of computation. This subsection presents our model transformation approach in detail.

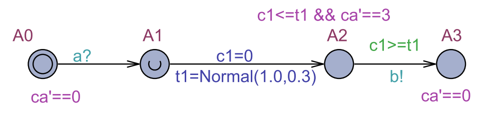
1) Preliminary Knowledge of NPTA: Unlike traditional timed automata, the clocks of a *Priced Timed Automaton* (PTA) [25] can evolve with different rates. To simplify the formal definition, we skip the introduction to the richer flavors of PTAs supported by UPPAAL-SMC, e.g., *urgent locations* [25]. Let C be a clock set. A *clock valuation* is a function $v : C \to R_{>=0}$ which maps C to the set of non-negative reals $R_{>=0}$. Let v_0 be the initial valuation where $v_0(c) = 0$ for all $c \in C$. Let $\mathcal{U}(C)$ ($\mathcal{L}(C)$) be the set of upper-bound (lower-bound) guards which are in the form $x \sim k$ or $x - y \sim k$, where $x, y \in C$, $k \in R$ and $\sim \in \{<, \leq, ==\}$ ($\sim \in \{>, \geq, ==\}$). Assuming $g \in \mathcal{L}(C) \cup \mathcal{U}(C)$, $v(C) \models g$ denotes that valuation $v(C)$ satisfies the constraint g. Definition 3.5 presents the formal definition of a PTA.

Definition 3.5: A *PTA* is a 8-tuple $A = (L, l_0, C, \Sigma, E, R, I, \tau)$ where: i) L is a finite set of locations; ii) $l_0 \in L$ is the initial location; iii) C is a finite set of clocks; iv) $\Sigma = \Sigma_i \cup \Sigma_o$ is a finite set of actions where Σ_i and Σ_o indicate exclusive input actions and output actions, respectively; v) $E \subseteq L \times \mathcal{L}(C) \times \Sigma \times 2^C \times L$ is a finite set of transitions, where $\mathcal{L}(C)$ denotes the transition guard and 2^C denotes a set of reset clocks of the transition; vi) $R : L \to F^C$ assigns a clock rate vector to each location, where $F(c)$ specifies the clock rate for $c \in C$ in the form of ordinary differential equations; vii) $I : L \to \mathcal{U}(C)$ assigns an invariant to each location; and viii) τ is the system clock which is never reset. ■

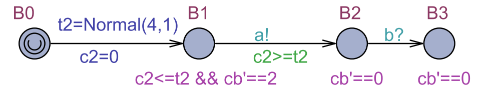
Let $A_i = (L^i, l_0^i, C^i, \Sigma^i, E^i, R^i, I^i, \tau^i)$ and $A_j = (L^j, l_0^j, C^j, \Sigma^j, E^j, R^j, I^j, \tau^j)$ ($i \neq j$) be two PTAs. The PTAs A_i and A_j are composable into a network only if $C^i \cap C^j = \emptyset$ and $\Sigma_o^i \cap \Sigma_o^j = \emptyset$. Definition 3.6 presents the formal structure of an NPTA.

Definition 3.6: Let $A_i = (L^i, l_0^i, C^i, \Sigma^i, E^i, R^i, I^i, \tau^i)$ ($1 \leq i \leq n$) be a PTA. Their composition ($A_1 \mid \ldots \mid A_n$) is an NPTA which is a 8-tuple $NA = (NL, NL_0, NC, N\Sigma, NE, NR, NI, \theta)$, where: i) $NL = \times_i L^i$; ii) $NL_0 = \times_i \{l_0^i\}$; iii) $NC = \cup_i C^i$; iv) $N\Sigma = \cup_i \Sigma^i$; v) $((l_1, \ldots, l_j, \ldots, l_n), g_j, a, r_j, (l_1, \ldots, l_j', \ldots, l_n)) \in NE$ whenever $(l_j, g_j, a, r_j, l_j') \in E^j$; vi) $NR(l_1 \ldots l_n)(x) = R^i(l_i)(x)$ when $x \in C^i$; vii) $NI(l_1, \ldots, l_n) = \wedge_i I^i(l_i)$ ($l_i \in L^i$); and viii) θ is the system clock shared by all the PTAs. ■

Let $(l, v) \in NL \times R_{>=0}^{NC}$ be an NPTA state where $l = (l_1, \ldots, l_n)$ is a composite of locations from different PTAs and $v \models NI(l)$. Let $v[X]$ indicate the reset operation on the clock set X. That is if $c \in X$, $v(c)$ is set to 0, otherwise $v(c)$ keeps its value. Following the composition rules [25], the semantics of an NPTA is mainly based on the following two kinds of transitions: i) a *discrete transition* $(l, v) \xrightarrow{a} (l', v')$ can be triggered if there is a transition (l, g, a, X, l') such that $v \models g$ and $v' = v[X]$; ii) a *delay transition* $(l, v) \xrightarrow{d} (l, v')$ can be triggered if $v' = v + \int_{v(\theta)}^{v(\theta)+d} NR(l)\, d\theta$ such that $v \models NI(l)$ and $v' \models NI(l)$, where $v(\theta)$ indicates the system time of entering state (l, v). Within an NPTA, PTAs communicate with each other using broadcast channels or shared variables. For more details about NPTA, please refer to [10].



(a) PTA *A*



(b) PTA *B*

Fig. 2. An NPTA ($A|B$)

2) An Example of NPTA with Uncertainties: Figure 2 shows an NPTA consisting of two PTAs A and B, where each PTA has four locations and two clocks (e.g., c_1 and c_a in A). The locations here marked with symbol "U" are urgent locations which can freeze time. In other words, time is not allowed to pass when a PTA is in an urgent location [10]. Note that clocks can evolve with different rates in different locations. The rate of a clock is 1 by default. To change the rate of a clock, we need to modify the rate value of the primed version of the clock. For example, $c_a' == 3$ in A_2 denotes that the rate of c_a is 3 in this location. Since UPPAAL-SMC supports the modeling of clock rates using expressions in the form of ordinary differential equations, UPPAAL-SMC can be used to model non-linear hybrid systems. Note that although UPPAAL-SMC only supports the uniform and exponential distributions explicitly, based on the C-like programming constructs and built-in function *random*() provided by UPPAAL-SMC various distributions can be constructed. For example, we can construct the normal distribution based on the Box-Muller approach [19]. Assume that the values of two variables t_1 and t_2 follow the normal distributions $N(1, 0.3^2)$ and $N(4, 1^2)$, respectively. The action t_1=*Normal(1,0.3)* on the outgoing edge of A_1 assigns t_1 with a random value following $N(1, 0.3^2)$. Since the invariant in A_2 is $c_1 \leq t_1$ and the guard on the outgoing edge of A_2 is $c_1 \geq t_1$, PTA A is stuck at A_2 for a time of t_1. If the NPTA is simulated for numerous times, the sojourn time at location A_2 follows the distribution $N(1, 0.3^2)$. In this example, PTAs are synchronized by two complementary action pairs ("!" indicating sending and "?" denotes receiving) via *urgent channels a* and *b*. While simulating $(A|B)$ with a large number of runs, we can find that the reaching time of the composite location $(A3, B3)$ follows the normal distribution $N(4+1, 1^2 + 0.3^2)$, since the sojourn time of $(A|B)$ at composite locations $(A0, B1)$ and $(A2, B2)$ follows the normal distributions $N(4, 1^2)$ and $N(1, 0.3^2)$, respectively. By adopting the NPTA template like the above example, arbitrarily complex stochastic behaviors can be modeled.

3) Mapping from Uncertain Hybrid AADL to NPTA: To facilitate the detailed modeling of system architectures, AADL provides more types of syntax modeling constructs than NPTA. Since our approach focuses on quantitative analysis of stochastic behaviors of Uncertain Hybrid AADL designs, during the model transformation we neglect all the AADL constructs which cannot affect the system behaviors. Table I shows the structural mappings from uncertain Hybrid AADL constructs to NPTA constructs. Note that we only list a subset of AADL constructs that have a strong correlation with uncertainty-aware hybrid features. This subset can essentially be used to fully describe the behaviors of hybrid systems within an uncertain environment. In our approach, we use the BLESS annex to specify the discrete components of systems, e.g., controller. We use the Hybrid annex to describe both discrete and continuous behaviors of hybrid components, e.g., plants. The instances of both annexes can be described using PTAs. Note that our proposed uncertainty annex focuses on the variation modeling of communication delays and parameter values. It only slightly changes structure of PTAs. To model the overall uncertain behaviors of the whole hybrid system, all the generated PTAs are synchronized through the channels, which are transformed from AADL connections. To guarantee the correctness of our mappings from AADL to NPTA and final translation results, we develop a comprehensive test suite which covers all possible notations of the three annexes. The testing results show that the NPTA generated by our transformation rules (templates) can correctly and accurately reflect the behaviors of Uncertain Hybrid AADL designs. In Section III-B5, we will show the details of such transformation rules with concrete examples.

TABLE I
CONSTRUCT MAPPINGS BETWEEN *AADL* AND *NPTA*

AADL Constructs	NPTA Constructs
system	$(PTA_1 \mid \ldots \mid PTA_n)$
thread / device	PTA template
(event) data port	variable
connection	urgent channel
property set / type	global variable
BLESS annex subclause	PTA template
Hybrid annex subclause	PTA template
Uncertainty annex subclause	PTA actions/invariants/guards

Our approach relies on the meta-models of AADL and its annexes to guide AADL model parsing as well as the construct mapping. Similar to the formal definitions presented in Section III-A, the meta-models define a set of correlated sub-constructs of the model, which can be used to extract the necessary information for the model transformation. Due to the space limitation, we do not introduce the meta-models used in our approach here. The meta-model for the AADL without annex can be found in [4], and the meta-models for BLESS and Hybrid annexes can be obtained from [16] and [6], respectively. Similar to BLESS and Hybrid annex, the meta-model of Uncertainty annex can be inferred from Definition 3.4.

In our approach, the generated NPTA model can be divided into two parts: i) *back-end configurations* that are used to declare necessary data structures (e.g., variables, channels) and functions (e.g., distributions, actions) for the stochastic modeling of NPTA modes, and ii) *front-end models* that are used to model the behaviors of hardware, software and environment components. Our approach assigns each of AADL components annotated by BLESS and Hybrid annex subclauses with a front-end model and a back-end configuration. Moreover, there is a global back-end configuration for the whole system whose information are shared by all the front-end models.

4) Back-end Configuration Generation: As a textual file, a back-end configuration mainly consists of: i) a set of declarations of variable and channels, and ii) a set of distribution and action functions. By using our approach, such information can be automatically extracted from AADL designs. Listing 2 shows the back-end configuration generated from the train AADL shown in Listing 1. To save the space, we put both the global configuration and the local configuration for the train within the same file.

For a back-end configuration, the global declarations of variable and channels are generated from the top level of AADL designs, i.e., hardware/software components and their interconnections. For each port of an AADL component, we create an *urgent channel*[1] which indicates the AADL *connection* associated with the port. For example, assuming that the connection name bound by the port *tv* in the AADL

[1]An urgent channel does not allow a delay if it is possible to trigger a synchronization over it. We only use urgent channels in our translation, since the timing behavior of NPTA using urgent channels is deterministic. We use urgent locations in the transformation for the same reason.

design is *_tv*, we will declare two things for this port in the global configuration. The first one is an urgent channel *c_tv* that is used for the synchronization with other components. Since *tv* is a data port for the velocity of trains, we declare a variable *v_tv* of type double to hold data value during the data transmission via the connection. The constants defined in the global NPTA configuration correspond to the constants (e.g., EOA) defined in top level of AADL designs. For each NPTA model, a clock *systime* is defined in the global configuration to model the system time. To model different stochastic behaviors of PTAs, the global back-end configuration comprises a library of distribution functions, which can be used by front-end NPTA models or the local configurations.

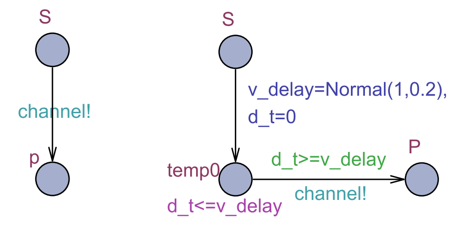
```
 //global declarations & distribution function lib.
urgent  chan c_tv, c_ts, c_ta;
double v_tv, v_ts, v_ta;
const double EOA=6000;
clock systime;
double Normal(double mean, double deviation){
// Box-Muller method
 }
double Uniform(double min, double max){
...
}
 //local configuration for Train.
clock s, v, a, t, d_t;
double  v_delay, fr;
void initialize(){
    fr=Normal(-0.1,0.05);
}
 //local configurations for other components.
 ...
```

Listing 2. Back-end configuration of *Train*

The local back-end configuration mainly deals with the definition of data structures for specific AADL components and annexes. During the model transformation, all the continuous AADL variables are converted to clock typed variables, and other AADL variables are converted to non-clock variables with different types. For each local configuration of PTAs, we define one built-in function *initialize()* which is used to initialize the values of variables. To enable the reset operation in a PTA, for each local configuration, we define one clock *d_t*. For the uncertainty annex in Listing 1, there are two variables declared, i.e., *v_delay* and *v_fr*. Note that only *v_delay* has a counterpart in the back-end configuration, since it is a time variable. The price variables have no counterparts, since they are used as intermediate variables during the model transformation. Note that the transformation rules for static and dynamic price variables are different. Since static price variables only take effect at the beginning of the simulation, we assign their random values to associated variables in the function *initialize()* of local back-end configurations. For example, we initialize the variable *fr* with a value following normal distribution $N(-0.1, 0.05^2)$ in the back-end function *initialize()* of the train example. Unlike static price variables, the dynamic price variables generate random values during the execution when necessary. It is widely used in front-end modeling to indicate the random value change of variables. For example, it can be used to model time-varying port delays.

5) Front-end Model Generation: As a graphical representation, front-end models are used in UPPAAL-SMC to describe the stochastic behaviors of PTAs. To describe the hybrid behaviors of systems, our approach adopts two kinds of annexes. We model the discrete behaviors of AADL components (e.g., thread component) using the BLESS annex. To describe continuous behaviors of components (e.g., device and abstract components), we use the Hybrid annex that is based on Hybrid CSP. For an AADL component without any annotated annexes, we assume a simplified semantics for its behavior, where the component periodically receives the data from its input ports and sends the data to its output ports. Therefore, the major task of front-end model generation is to transform uncertainty-aware BLESS and Hybrid annexes to their NPTA counterparts.

Uncertainty Modeling of Front-end Model: For front-end model transformation, we consider two kinds of uncertainties in the Uncertainty annex. The first one is described by time variables which are used to model the delay variations of network communication or task execution. To model such stochastic timing behaviors of system, we use the transformation pattern as shown in Figure 3. Figure 3(a) shows a scenario that the PTA tries to send something via the channel using the action *channel!*. Without annotated Uncertainty annexes, the sending time of the action is fixed. However, by using our Uncertainty annex, we can associate a time variable *v_delay* following normal distribution (i.e., $N(1, 0.2^2)$) with this channel. By splitting the transition in Figure 3(a) and introducing a temporary location to indicate the waiting, we can model the scenario that the action time follows the distribution $N(1, 0.2^2)$ as shown in Figure 3(b). On the incoming edge of the newly added location *temp*, we assign *v_delay* with a random value following $N(1, 0.2^2)$ and we reset the clock *d_t*. Note that the invariant of location *temp* is *d_t<=v_delay* and the guard on the outgoing edge of *temp* is *d_t>=v_delay*. Therefore, the PTA remains at location *temp* for a time of *v_delay* following the normal distribution $N(1, 0.2^2)$. The second uncertainty in the front-end model is specified by dynamic price variables in the uncertainty annex, which mimics the random values of parameters (e.g., sensor inputs). The transformation of such uncertainty only needs to assign or replace the applied variable with the given distribution function.



(a) Certain (b) Uncertain

Fig. 3. Transformation pattern for uncertain PTA models

BLESS Annex Based PTA Generation: Listing 3 presents an AADL example annotated with both BLESS and Uncertainty annex instances, which describes the uncertain behaviors of the controller component within the CTCS-3 Movement Authority scenario (see details in Section IV). Meanwhile, Figure 4 shows the corresponding PTA generated using our transformation rules. Due to the space limitation, we only give the partial AADL specification and corresponding PTA for the controller.

```
thread Controller
    features
    r: out event port;
    m: in event data port CTCS_Types::MovementAuthority;
    ea: in data port CTCS_Types::EOA;
    properties
    Dispatch_Protocol => Periodic;
    Period => 200 ms;
end Controller;

thread implementation Controller.impl
annex BLESS{**
    variables
    e   : CTCS_Types::EOA;
    iMA : CTCS_Types::MovementAuthority:=null;
    states
    READY : initial state;
    GMA   : complete state;
    CMA   : state;
    RETRY : state;
    MFR   : complete state;
    -- other state declarations
    transitions
    T0_go: READY -[]-> GMA {r!};
    T1_MA_Check: GMA-[on dispatch]->CMA{m?(iMA);ea?(e)};
    T2_MA_Ok: CMA -[not (iMA=null)]-> MFR{ };
    T3_MA_NotOk: CMA-[iMA=null]-> RETRY {};
    T4_MA_Retry: RETRY -[]-> GMA {r!};
    -- other transition declarations
**};

annex Uncertainty {**
    variables
    time t_delay applied to Controller.r
    distributions
    t_delay = Normal(0.2,0.07)
**}
end Controller.impl;
```

Listing 3. Uncertain AADL Design for *Controller* in CTCS-3 MA

From this example, we can find that the BLESS annex shares a large overlap of the modeling constructs with PTA. In BLESS annex, there are four kinds of states: *initial state*, *complete state*, *execution state*, and *final state*. Based on the definition [16], the *initial state* and *execution state* (with keyword "state") are only used for the initialization and branch operation, respectively. Therefore, they are converted into urgent states in corresponding PTA, since they do not consume any time. Conversely, the state machine of BLESS annex leaves *complete state* upon periodical dispatch events, and will stay at the *final state* forever when the whole controller task finishes. Therefore, these two kinds of states will be directly mapped to general PTA locations. Since the definition of BLESS transitions is more complex than the one of PTA transitions, we need to use some specific transformation pattern to generate PTA counterparts with equivalent semantics. In BLESS annex, a transition allows for a sequence of send (denoted by "!") and receive (denoted by "?") actions. As an example shown in Listing 3, the transition *T1_MA_Check* has two receive actions, i.e., *m?(iMA)* and *ea?(e)*. Therefore, after staying in the state *GMA* for 200 milliseconds (indicated by the guard *dispatch>=0.2*), the machine starts to receive the message from the input event port *m* and input data port *ea* in a sequential order. However, PTA models only allow one such action on a transition. To model such combined send and receive actions, for each action on the BLESS transition, we introduce one auxiliary transition and one temporary location to trigger the action in the order of their occurrences on the BLESS transition. For example, in Figure 4 two temporary locations (*temp1* and *temp2*) are introduced for the two consequent receive actions. Moreover, BLESS annex provides the *assert* section and *invariant* section to specify the behavior constraints of an AADL component. Such information can be directly parsed and used as the transition guards and location invariants in the generated PTA models. Since the AADL design of the controller does not contain such information, we do not show the corresponding translation in Figure 4.

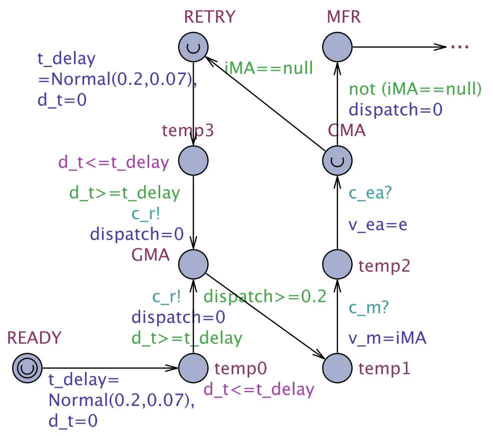


Fig. 4. PTA model of *Controller* (partial)

Note that, to model the network delay, in Uncertainty annex there may be some time variables associated with the ports/channels used by the send actions. In this case, for each send action we need to set the delay information on the newly introduced location and transition pair. For example, in the Uncertainty annex of Listing 3, the time variable *t_delay* following the normal distribution $N(0.2, 0.07^2)$ is applied to the port *r* of the controller. Since there are two transitions (*T0_go* and *T4_MA_Regry* in Listing 3) that conduct the send action, we need to set the delay information on both the PTA locations *temp0* and *temp3* and their outgoing transitions using the transformation template as shown in Figure 3.

Hybrid Annex Based PTA Generation: When transforming a Hybrid annex annotated AADL [6], the front-end PTA model is mainly extracted from the *behavior* section. Since Hybrid annex adopts the process algebra notations, the behavior of component is described by a set of CSP *process*, e.g., *Train* defined in the hybrid behavior section of Listing 1.

During the transformation, each CSP process is converted to a location except for the *skip* process (e.g., *Continue* in Listing 1). The continuous evolution of a CSP process is expressed using differential expressions, which are translated and used as the invariant of the corresponding non-urgent location. As an example shown in Listing 1, the differential expression *'DT 1 s=v'* indicates that the derivative of *s* is *v*. It can be translated into the derivative expression *s'=v* and used as a part of invariant for the location *Train*.

To enable the communication between computation components and physical environments, the semantics of Hybrid AADL allows two kinds of interruptions, i.e., *timed interrupts* and *communication interrupts*. In the behavior section of Hybrid annex, a *timed interrupt* is defined as a part of CSP process in the form of $[> time\_val] >$, which will preempt the continuous evolution after an amount of time (i.e., *time_val*). By using the similar transformation pattern shown in Figure 3, we can assure that the continuous evolution of CSP process can be interrupted after a time of *time_val*. The *communication interrupts* enable the preemption of continuous evolutions by communication events via AADL ports. For example, the communication interrupt in the form of $[[> pout!(v)]] > EV$ denotes that whenever a value of v is sent out the port *pout*, the current evolution will be terminated and the CSP process *EV* will be adopted as the subsequent behavior of the process.

Generally, a communication interrupt may contain a sequence of send or receive actions. During the PTA transformation, we model the actions based on their occurrence order in the interrupt. For each action, we generate a new PTA location together with a new transition with the corresponding action on it. Note that in the generated PTA a location should be set as urgent if the action on next edge is a send action. When translating the *choice* operator of a CSP process, we will create a new adjacent location in the generated PTA for each alternative. The subsequent behavior of the process is determined by the Boolean expression associated with the alternative. To model the behavior of a repeating process defined in the behavior section, we connect the last location to the first location of the process in the PTA to form a loop.

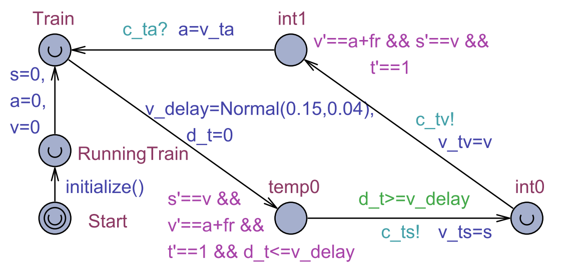


Fig. 5. PTA model of *Train*

Figure 5 shows the PTA of the Train example defined in the behavior section of Listing 1. To enable the execution of function *initialize()* defined in the local back-end configuration, we introduce an urgent location *start*. On the outgoing transition of location *RunningTrain*, the continuous variables s, a and v are initialized. Since the first action in the communication interrupt is a send action (i.e., *ts!(s)*), we make the location *Train* urgent. As defined in the uncertainty annex, the channel associated with the port *ts* has a communication delay following $N[0.15, 0.04^2]$. Based on the pattern shown in Figure 3, we need to create a new location (i.e., *temp0*) to model the delay information. Note that since *ts* is a data port, we need to sent the value of s via this port. However, the corresponding action *c_ts!* on the outgoing edge of *temp0* cannot hold the value information. Therefore we use the variable *v_ts* which corresponds to the channel *c_ts* to hold the data value during the communication via the channel. Since an urgent location allows no invariants, we move the invariant derived from the different expression of CSP process *Train* to the new location *temp0*. Since there are three actions in the communication interrupt of the CSP process *Train*, we create two new locations to perform the actions according to their occurrence order. Note that the newly introduced three locations (i.e., *temp0*, *int0* and *int1*) can be considered as the sub-locations of CSP process *Train*. Therefore, they should have the same location invariant. For the action *ta?(a)* of the communication interrupt, we need to get the data value from port *ta*. Therefore, we use the action *a=v_ta* to update the value of *a*.

C. Property Generation for Quantitative Analysis

To enable the quantitative evaluation of Uncertain Hybrid AADL designs, our proposed Uncertainty annex allows designers to specify design requirements as performance queries. These performance queries will be transformed as properties in the form of cost-constrained temporal logic to reason the performance of the NPTA models generated from Uncertain Hybrid AADL designs. Since we focus on the reasoning of stochastic behaviors of AADL systems, the designers can conduct following two kinds of queries.

- **Performance query:** The performance query can be used to check the probability that an expected performance metric can be achieved under a given resource limit. The performance metric can be expressed as the predicate and the resource limit can be specified as the constraint using the keyword *under*.
- **Safety query:** The safety query can be used to check the probability that an unexpected scenario can happen eventually with a given resource limit. In the query, the unexpected scenario can be expressed as the predicate and the resource limit can be specified as the constraint using the keyword *under*.

Although safety queries and performance queries have different meaning, they share the same template during the property generation. In the *queries* section, a query consists of two parts, i.e., predicate ϕ and resource constraint ψ. The predicate ϕ can be used to denote either an unexpected scenario or an expected performance metric.

To evaluate the performance of generated NPTA models, UPPAAL-SMC adopts cost-constrained temporal logic [11] based performance queries in the form of $Pr[bound](<> expr)$, where $[bound]$ indicates the bound of the cost and the expression $<> expr$ asserts that the scenario *expr* should happen eventually. By using our approach, the queries will be transformed into properties in the form of $Pr[\psi](<> \phi)$. For example, the performance query $p2$ in Listing 1 intends to check the probability that the travel length of the train exceeds 4 kilometers within 200 seconds. In order to conduct the quantitative evaluation using UPPAAL-SMC, the query will be converted to a property $Pr[<= 200](<> Train.ts >= 4000)$ in the form of cost-constrained temporal logic. Based on the specified *probability of false negatives* (i.e., α) and *probability uncertainty* (i.e., ε), UPPAAL-SMC will simulate a specific number of stochastic runs which are terminated when either *bound* or $<> expr$ holds. The success rate p of $<> expr$ satisfying *bound* will be reported in the form of a probability range $[p-\varepsilon, p+\varepsilon]$ with a specified confidence $1-\alpha$.

IV. Case Study

To show the efficacy of our approach in analyzing system performance within uncertain environments, this section presents the experimental results of verifying the *Movement Authority* (MA) control of *Chinese Train Control System Level 3* (CTCS-3) [20], [28]. By using our proposed Uncertainty annex, we extended the hybrid CTCS-3 AADL model presented in [20] using the tool OSATE2 [7]. The uncertainty information in the model is suggested by railway experts from our industrial partner Casco Signal Ltd. Based on our *XMI parser* and *NPTA model generator* implemented using JAVA[2], we can obtain the corresponding NPTA model as well as performance queries. We employed the model checker UPPAAL-SMC (version 4.1.19, $\alpha = 0.02$, $\varepsilon = 0.02$) to conduct the evaluation. All the experimental results were obtained on a desktop with 3.3GHz AMD CPU and 12GB RAM.

A. System Model of CTCS-3 MA Scenario

As one of the fourteen basic scenarios of CTCS-3 *System Requirements Specification* (SRS), the MA control plays an important role in prohibiting trains from colliding with each other. Typically an MA scenario involves three major components as follows: i) *trains* that periodically (every 500 milliseconds) send their status information (i.e., current location and velocity) to the *controller* and receive acceleration information directed by the *controller*; ii) *Radio Block Centers (RBCs)* that provide MAs to trains based on information exchange with trackside subsystems and the on-board *controller*; and iii) on-board *controller* subsystems which control the velocity of trains by changing their accelerations.

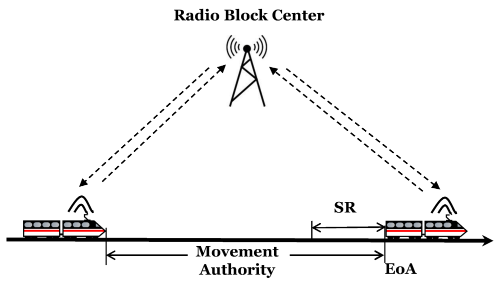


Fig. 6. MA scenario of CTCS-3 [20]

As shown in Figure 6, the RBC assigns a dynamic MA to the left train based on the track situation and the movement of the right train. Here, EOA stands for the *End of Authorization*. When a train reaches a specific distance (i.e., SR) away from EOA, it needs to apply for a new MA. If the authorization is not granted in time, according to SRS the train should stop before the EOA. According to SRS [28], an MA comprises a sequence of *segments*, where each *segment* has two speed limits v_1 and v_2 ($v_1 \geq v_2$). In this example, we set the speed limits v_1 and v_2 for each segment to $73m/s$ and $66m/s$, respectively. If the train speed exceeds v_1 (v_2), an emergency (normal) brake will be performed to slow down the train. Upon receiving an MA request from *controller*, RBC will reply a new MA together with all the segment information (e.g., speed limits, operation mode). More details can be found in [20], [28]. In this example, we set the length of an MA to 6 kilometers, and set the length of SR to 1 kilometer. The train starts with a speed of $0m/s$. All the *segments* have the same length and speed limits. Note that the SRS requirements cannot be guaranteed within an uncertain environment. For example, due to the mutual interference between varying communication delays and friction coefficient of tracks, inaccurate estimation of locations can make the train pass the EOA. Although train drivers can perform emergency brake manually, proper quantitative analysis of these unsafe scenarios should be studied at architecture level to make the train movement more safe.

[2]We have shared our tool (including the source code of Uncertain Hybrid AADL parser and NPTA model generator) and the uncertain CTCS-3 MA example on Github. The download address is https://github.com/tony11231/aadl2uppaal.

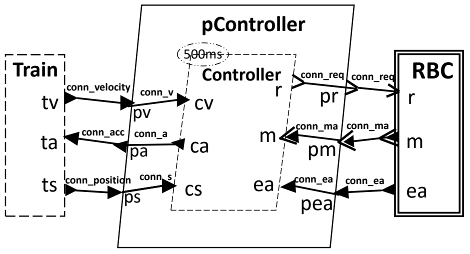


Fig. 7. AADL model for CTCS-3 MA

Figure 7 shows the graphical AADL model for CTCS-3 MA design, where the *controller* plays a central role. Within the MA scenario, the *controller* sends the MA request to RBC via the port *r* and receives the segment and EOA information from the ports *m* and *ea*, respectively. To achieve the train status, the *controller* receives the location and speed information from the ports *cs* and *cv* every 500 milliseconds. It also controls the train by specifying the newly calculated acceleration for the train via port *ca*. Although this figure does not explicitly present any uncertainty information, in this example we consider various uncertainties that may affect the performance of the MA control, e.g., communication delays between RBC and controllers, computation time variations of both software/hardware components of controllers, and varied coefficient of friction of tracks. The cumulative variations by all these uncertainties strongly affect the performance and safety of the CTCS-3 MA. In other words, the risk of train collisions is high within an uncertain environment.

B. Uncertainties in CTCS-3 MA Scenario

As shown in Table II, this experiment took nine uncertain aspects of CTCS-3 MA into consideration. Similar to the work in [21], [26], this paper adopts normal distributions to model the performance variations in the CTCS-3 MA scenario. All such variation information was collected from historical data of train operations. Note that our approach supports a variety of distributions, which can be used to accurately model the Uncertain Hybrid AADL designs. In this table, the first column presents the category of the uncertainties. The second column presents the AADL constructs that cause the uncertainties. For example, when *controller* sends a MA request to RBC via port *Controller.r*, there is a delay variation caused by the connection *conn_req* following the distribution $N(0.1, 0.03^2)$, where the expected execution time is 0.1 seconds and the standard deviation is 0.03 seconds. Note that during the statistical model checking the network delay of 0.1 seconds with standard deviation of 0.03 may lead to a negative value. In our approach, if the variable with type "time" is randomly assigned with a negative value, we will set it to 0. According to the three-sigma rule, this approximation will still be accurate in this case. The last two columns provide the variation distributions and value unit, respectively. By using our tool chain, the NPTA model of the Uncertain Hybrid AADL design can be obtained automatically.

TABLE II
UNCERTAINTIES OF MA COMPONENTS

Causes	Constructs	Variations	Unit
Network Delay	*Controller.r*	$N(0.1, 0.03^2)$	Seconds
	RBC.m	$N(0.1, 0.03^2)$	Seconds
	RBC.ea	$N(0.1, 0.03^2)$	Seconds
	Train.tv	$N(0.15, 0.04^2)$	Seconds
	Train.ts	$N(0.15, 0.04^2)$	Seconds
	Controller.ca	$N(0.17, 0.04^2)$	Seconds
Parameter	*Train.fr*	$N(-0.1, 0.05^2)$	MPSS*
Execution Time	*RBC.T0*	$N(0.1, 0.03^2)$	Seconds
	Controller.T5	$N(0.2, 0.07^2)$	Seconds

**MPSS* indicates *Meter Per Second Squared.*

C. Performance Analysis for CTCS-3 MA Scenario

To focus on quantitative analysis of the MA scenario influenced by uncertain factors, we investigated stochastic behaviors of a train within an MA as shown in Figure 6. We assume that the train will fail to get the next MA when entering SR. Therefore, it should stop before EOA. By using our tool, three queries are generated to analyze the performance of Uncertain Hybrid AADL design for CTCS-3 MA.

To investigate the probability that a train can stop safely before the end of authorization within 300 seconds, we adopt the performance query $Pr[<= 300](<> Train.v <= 0 \;\&\&\; Train.s < 6000 \;\&\&\; Train.s > 0)$, where $Train.v$ denotes velocity of the train and $Train.s$ indicates the location of the train. Figure 8 presents the evaluation results for the query in the form of Cumulative Probability Distribution (CPD). In this figure, the x-axis denotes the time limit, and the y-axis indicates success rate of the performance requirement indicated by the query. In this evaluation, we considered three Uncertain Hybrid AADL designs, where the accelerations directed by the controller are different. We set the accelerations of three designs to 0.3 MPSS (Meter Per Second Squared), 0.4 MPSS and 0.7 MPSS, respectively. By running 868 runs, we can get a probability interval [0.91,0.95] with a confidence 98% for the query of the AADL design with an acceleration of 0.3 MPSS. The SMC simulation for this query costs around 132 seconds. For the AADL designs with acceleration of 0.4 MPSS and 0.7 MPSS, we can get probability intervals [0.88,0.92] and [0.81,0.85] with a confidence of 98%, respectively. From this figure, we can find that the CPD of the design with 0.7 MPSS rises earlier (i.e., 173 seconds), since it has a larger acceleration and can reach the speed limit v_2 more quickly than the other two designs. However, the larger acceleration indicates the higher difficulty in managing the train speed. In other words, the chance that the train exceeds EOA becomes higher. Therefore, we can find that the AADL design with 0.3 MPSS can achieve the highest success rate to stop before reaching EOA. Moreover, we can find that the success rate will not increase significantly after a time threshold, since the train has stopped before the time limit, i.e., 300 seconds.

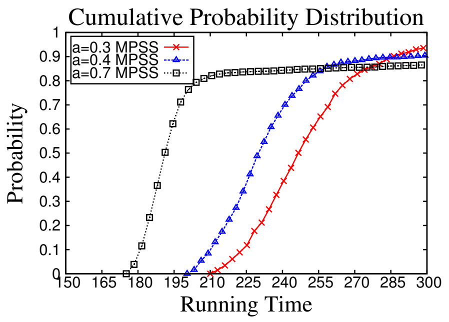


Fig. 8. Performance query results with different accelerations

The interaction frequency between trains and the controller plays an important role in CTCS-3 MA design, since it strongly affects the cost and performance of train designs. Although longer control periods cost less communication bandwidth, the infrequent updates of train accelerations make the train hard to be controlled. To investigate the effects of different control periods, we assume that the acceleration (without consider frictions) sent from the controller is fixed (i.e., 0.4 MPSS) for the train design. Figure 9 shows the evaluation results of using the same query as the one used in Figure 8. We consider three designs with different control periods, i.e., 0.2, 0.5 and 0.7 seconds, respectively. From this figure, we find that the design with the smallest control period (i.e., 0.2S) can achieve the highest rate of success. By running 266 runs, we can achieve a probability interval [0.95,0.99] with a confidence 98% for the query of the AADL design with a control period of 0.2 seconds.

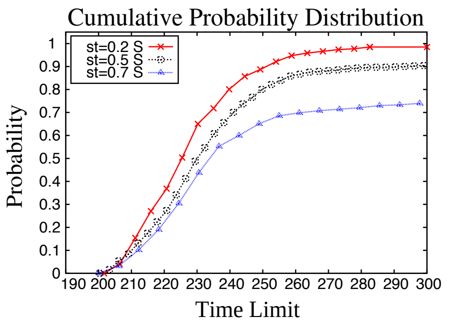


Fig. 9. Performance query results with different control periods

To determine the performance of the AADL design, we used the query $Pr[<= 200](<> Train.s >= 4000)$ which checks whether the train can run a distance of 4.0 kilometers within 200 seconds. As shown in Figure 10, we adopted three designs with different accelerations. We can find that the performance difference among these three designs is quite small. The design with an acceleration of 0.3 MPSS achieves the worst performance, since it needs a worst-case time of 193 seconds to reach the specified location. Interestingly, the design with 1.0 MPSS does not win the comparison. It needs longer time to hit the specified location than the design with 0.6 MPSS, since the design with a larger acceleration will have a more drastic speed update near the speed limits.

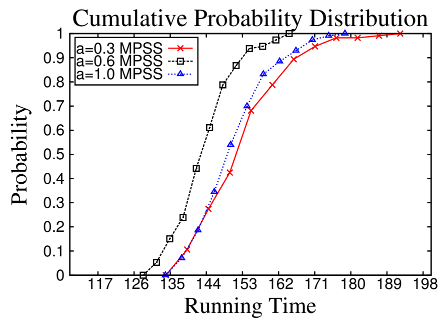


Fig. 10. Performance query results for reaching a location

From the above experimental results, we can find that our approach can be used to effectively reason about the performance of designs within complex uncertain environments. Our approach can not only support the quantitative evaluation of specified design using performance queries, but also can be used for the purpose of design optimization based on parameter tuning.

D. *Quantitative Safety Analysis for CTCS-3 MA Scenario*

During the running of the train, we expect the train speed not to exceed the upper speed limit v_1, since it can easily make the train derailed. Therefore, when the train reaches the speed v_1, we need to apply the urgent brake to reduce the train speed drastically. To check the probability of overspeed of trains, we used the safety query in the form of $Pr[Tran.s <= 5000](<> Train.v >= 73)$, which indicates that within a distance of $|EOA - SR|$ the train speed cannot be larger than or equal to v_1 (i.e., $73m/s$).

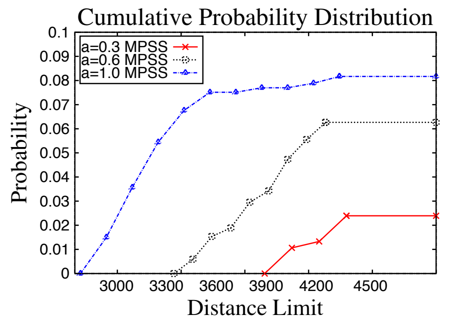


Fig. 11. Safety query results for overspeed

Figure 11 shows the evaluation results for the three AADL designs with different accelerations. For the design with an acceleration of 0.3 MPSS, UPPAAL-SMC uses 17 seconds to obtain a probability interval [0.009, 0.049] for the query. From this figure, we can find that the larger the acceleration is, the higher chance the train can exceed the upper speed limit. To achieve a 2% chance of overspeed, the design with 1.0 MPSS needs an average travel distance of 3.0 kilometers, whereas the designs with 0.3 MPSS and 0.6 MPSS need an average of 3.5 kilometers and 4.1 kilometers, respectively. From the above evaluation results generated by our approach, we can clearly figure out the safety information for the designs within uncertain environment. Based on the comparison among designs with different parameter values, we can achieve reasonable design settings under a given safety requirement.

V. CONCLUSIONS

This paper proposed a novel SMC-based framework that enables quantitative performance evaluation of Hybrid AADL designs considering various uncertain factors caused by physical environments. We introduced a lightweight language extension to AADL called Uncertainty annex for the stochastic behavior modeling. By using our proposed transformation rules, the uncertainty-aware Hybrid AADL designs can be automatically converted into NPTA models. Based on the statistical model checker UPPAAL-SMC, our framework enables automated evaluation of Uncertain Hybrid AADL designs against various complex performance and safety queries. Comprehensive experiment results carried on the CTCS-3 MA scenario demonstrate the efficacy of our approach.

REFERENCES

[1] E. A. Lee, "Cyber Physical Systems: Design Challenges", in *Proc. of Int. Symp. on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363–369.

[2] J. Delange and P. H. Feiler, "Incremental Latency Analysis of Heterogeneous Cyber-Physical Systems", in *Proc. of Int. Workshop on Real-Time and Distributed Computing in Emerging Applications (REACTION)*, 2014, pp. 21–27.

[3] P. H. Feiler and D. P. Gluch, "Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language", *Addison-Wesley*, 2012.

[4] SAE Aerospace, "SAE AS5506B: Architecture Analysis & Design Language (AADL) Standard Document", *SAE International*, 2012.

[5] J. Delange and P. H. Feiler, "Architecture Fault Modeling with the AADL Error-Model Annex", in *Proc. of EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA)*, 2014, pp. 361–368.

[6] 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", in *Proc. of ACM Annual Conference on High Integrity Language Technology (HILT)*, 2014, pp. 29–38.

[7] OSATE2, *http://osate.github.io/*.

[8] K. Hu, T. Zhang, Z. Yang, and W.-T. Tsai, "Exploring AADL Verification Tool Through Model Transformation", *Journal of Systems Architecture*, vol. 61, no. 3, pp. 141–156, 2015.

[9] A. Johnsen, K. Lundqvist, P. Pettersson, and O. Jaradat, "Automated Verification of AADL-Specifications using UPPAAL", in *Proc. of Int. Conf. on High-Assurance Systems Engineering (HASE)*, 2012, pp. 130-138.

[10] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen, "UPPAAL SMC Tutorial", *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 17, no. 4, pp. 1–19, 2015.

[11] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and Z. Wang, "Time for Statistical Model Checking of Real-Time Systems", in *Proc. of Int. Conf. on Computer Aided Verification (CAV)*, 2011, pp. 349–355.

[12] M. Bozzano, A. Cimatti, J. Katoen, V. Y. Nguyen, T. Noll, M. Roveri, and R. Wimmer, "Safety, Dependability and Performance Analysis of Extended AADL Models", *The Computer Journal*, vol. 54, no. 5, pp. 754–775, 2011.

[13] H. Yu, Y. Ma, T. Gautier, L. Besnard, P. L. Guernic, and J. P. Talpin, "Polychronous Modeling, Analysis, Verification and Simulation for Timed Software Architectures", *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1157–1170, 2013.

[14] K. Bae, P. C. Ölveczky, and J. Meseguer, "Definition, Semantics, and Analysis of Multirate Synchronous AADL", in *Proc. of International Conference on Formal Methods (FM)*, 2014, pp. 94–109.

[15] E. Jahier, N. Halbwachs, P. Raymond, X. Nicollin, and D. Lesens, "Virtual Execution of AADL Models Via a Translation into Synchronous Programs", in *Proc. of International Conference on Embedded Software (EMSOFT)*, 2007, pp. 134–143.

[16] B. R. Larson, P. Chalin, and J. Hatcliff, "BLESS: Formal Specification and Verification of Behaviors for Embedded Systems with Software", in *Proc. of NASA Formal Methods*, 2013, pp. 276–290.

[17] L. Zou, J. Lv, S. Wang, N. Zhan, T. Tang, L. Yuan, and Y. Liu, "Verifying Chinese Train Control System Under a Combined Scenario by Theorem Proving", in *Proc. of Verified Software: Theories, Tools, Experiments (VSTTE)*, 2013, pp. 262–280.

[18] E. Clarke, O. Grumberg and D. Peled. *Model Checking*. MIT Press, 1999.

[19] G. Box, and M. E. Muller, "A Note on the Generation of Random Normal Deviates", *The Annals of Mathematical Statistics*, vol. 29, no. 2, pp. 601–611, 1958.

[20] E. Ahmad, Y. Dong, B. R. Larson, J. Lv, T. Tang, and N. Zhan, "Behavior Modeling and Verification of Movement Authority Scenario of Chinese Train Control System Using AADL", *Science China Information Sciences*, vol. 58, no. 11, pp. 1–20, 2015.

[21] M. Chen, D. Yue, X. Qin, X. Fu, and P. Mishra, "Variation-Aware Evaluation of MPSoC Task Allocation and Scheduling Strategies using Statistical Model Checking", in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 199–204.

[22] F. Gu, X. Zhang, M. Chen, D. Große, and R. Drechsler, "Quantitative Timing Analysis of UML Activity Diagrams Using Statistical Model Checking", in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 780–785.

[23] M. Chen, S. Huang, X. Fu, X. Liu, and J. He, "Statistical Model Checking-Based Evaluation and Optimization for Cloud Workflow Resource Allocation", *IEEE Transactions on Cloud Computing*, accepted.

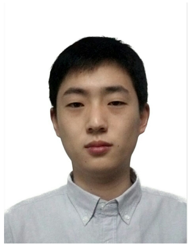
[24] H. Bruintjes, J. Katoen, and D. Lesens, "A Statistical Approach for Timed Reachability in AADL Models", in *Proc. of International Conference on Dependable Systems and Networks (DSN)*, 2015, pp. 81–88.

[25] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. Vliet, and Z. Wang. "Statistical Model Checking for Networks of Priced Timed Automata", in *Proc. of Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2011, pp. 80–96.

[26] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 1, pp. 8–23, 2013.

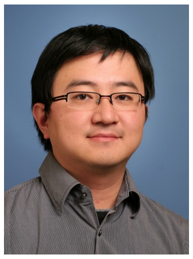
[27] P. J. L. Cuijpers and M. A. Reniers, "Hybrid Process Algebra", *The Journal of Logic and Algebraic Programming*, vol. 62, no. 2, pp. 191–245, 2005.

[28] The Ministry of Railways of The People's Republic of China, "System Requirements Specification of the CTCS-3 Train Control System", *Beijing: China Railway Publishing House*, 2008.

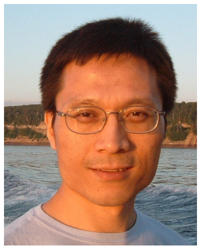


Yongxiang Bao received the B.E. degree from the Department of Computer Science and Technology, Anhui University of Technology, Anhui, China, in 2014. He is currently a master student in the Department of Embedded Software and System, East China Normal University, Shanghai, China. His research interests are in the area of design automation of embedded systems, statistical model checking, and software engineering.

Mingsong Chen (S'08-M'11) received the B.S. and M.E. degrees from Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006 respectively, and the Ph.D. degree in Computer Engineering from the University of Florida, Gainesville, in 2010. He is currently a Professor with the Computer Science and Software Engineering Institute at East China Normal University. His research interests are in the area of design automation of cyber-physical systems, formal verification techniques and cloud computing. He is an Associate Editor of IET Computers & Digital Techniques, and Journal of Circuits, Systems and Computers.



Qi Zhu (M'12) is an Assistant Professor of Electrical and Computer Engineering at the University of California, Riverside (UCR). He received his B.E. degree in Computer Science from the Tsinghua University, China in 2003, and his Ph.D. degree in Electrical Engineering and Computer Sciences from the University of California, Berkeley in 2008. Prior to joining UCR, He was a research scientist at the Strategic CAD Labs in Intel from 2008 to 2011. His research interests include model-based design and software synthesis for cyber-physical systems, CPS security, energy-efficient buildings and infrastructures, and system-on-chip design. He is a recipient of the 2016 CAREER award from the National Science Foundation, and best paper awards of ACM Transactions on Design Automation of Electronic Systems 2016, International Conference on Cyber-Physical Systems 2013, Design Automation Conference 2007 and 2006.



Tongquan Wei (S'06-M'11) received his Ph.D. degree in Electrical Engineering from Michigan Technological University in 2009. He is currently an Associate Professor in the Department of Computer Science and Technology at the East China Normal University. His research interests are in the areas of green and reliable embedded computing, cyber-physical systems, parallel and distributed systems, and cloud computing. He serves as a Regional Editor for Journal of Circuits, Systems, and Computers since 2012. He also served as Guest Editors for several special sections of IEEE TII and ACM TECS.

Frederic Mallet (M'01) is a Professor of Computer Science in Université Nice Sophia Antipolis. He works on the definition of sound models and tools for the design and analysis of embedded systems and cyber-physical systems. He is a permanent member of the Aoste team, a joint team between Inria Sophia Antipolis research center and I3S Laboratory (Cnrs UMR). During several years, he has been a voting member of the OMG Revision Task Forces for MARTE and SysML, where he was leading the definition of the allocation subprofile and had a key role in the definition of MARTE Time Model and MARTE/CCSL. He has also contributed to the working group between MARTE RTF and AADL committee.

Tingliang Zhou received his B.E. degree from Tongji University, China in 2012, and M.E. degree from Shanghai Jiaotong University, China in 2005 – all in computer science. He is currently a senior engineer and division manager of the Casco Signal Ltd., Shanghai, China. His research interests are in the area of trustworthy design of communication-based train control system, and formal verification techniques.