Quantitative Timing Analysis for Cyber-Physical Systems Using Uncertainty-Aware Scenario-Based Specifications

Ming Hu, *Member, IEEE*, Wenxue Duan, Min Zhang, *Member, IEEE*, Tongquan Wei, *Senior Member, IEEE*, and Mingsong Chen, *Senior Member, IEEE*

***Abstract*—Due to the merits of intuitive and visual modeling of design requirements, unified modeling language (UML) sequence diagrams are widely used as scenario-based specifications in the design of cyber-physical systems (CPSs). However, when more and more CPS products are deployed within an uncertain environment, existing sequence diagram analysis approaches cannot be used to accurately capture and quantify their timing behaviors at an early design stage. To address this problem, this article extends UML sequence diagrams to allow the modeling of stochastic system inputs, message processing time, and network delays, which strongly affect the system timing behaviors. We develop a statistical model checking-based framework that can automatically convert stochastic sequence diagrams into networks of priced timed automata to enable the quantitative analysis under various performance queries. The experimental results of two industrial designs in the railway field demonstrate the effectiveness of our approach.**

***Index Terms*—Cyber-physical systems (CPSs), quantitative timing analysis, scenario-based specification, unified modeling language (UML) sequence diagrams.**

I. Introduction

Along with the increasing popularity of cyber-physical systems (CPSs), the design complexity of software systems is skyrocketing due to the frequent interactions with external environment [1]–[3]. As a result, how to model the stochastic timing behaviors of software systems within uncertain environments and ensure both expected functional and nonfunctional (i.e., real-time, quality of service) requirements of specifications are becoming a major challenge in complex CPS design [4]–[7].

As a promising modeling specification, unified modeling language (UML) sequence diagrams [8], [9] are widely used by requirements engineers and domain experts to specify the interactions among different environment and system entities [10], [11]. By describing design requirements in an intuitive and visual manner, UML sequence diagrams focus on message exchanges among different communicating objects in distributed software systems. Although standard UML sequence diagrams are promising in analyzing the temporal order of message flows, they can hardly specify real-time constraints between the events of message sending and receiving. The situation becomes even worse when message execution time and network delay cannot be predicted within an uncertain environment [12], [14].

To guarantee the correctness and performance of UML sequence diagrams, a bunch of model checking-based approaches have been proposed to conduct the reachability analysis, constraint conformance analysis, and bounded delay analysis [13]. However, most of them can only answer "yes" or "no" for given safety properties. Few of existing UML sequence diagram-based approaches can model and evaluate the stochastic timing behaviors of software systems involving various uncertain factors [15]. When taking uncertain environment into account, most designers are more concerned about "the probability that an expected scenario can be fulfilled within a given time limit." Unfortunately, due to random system inputs, message processing time, and network delays, it is hard to figure out the answer and explain how to improve the system performance. Although probability-based approaches [16], [17] can be used to model various kinds of time variations, few of them can accurately model the parallel execution as well as alternative decisions supported by UML sequence diagrams. Apparently, the bottleneck here is the lack of stochastic semantics as well as effective quantitative analysis methods for UML sequence diagrams.

Relying on monitoring random simulation runs of target systems and the analyses using statistical methods (e.g., Monte Carlo simulation and sequential hypothesis testing). Statistical model checking (SMC) [18] is a promising approach in estimating the satisfaction probability of user-specified performance queries. Since SMC is not fully based on formal verification, it requires far less checking time than traditional formal verification techniques. Therefore, it can be used to validate complex CPS designs against various timing-oriented performance queries. In our approach, we use the statistical model checker UPPAAL-SMC [19], [20] as the engine

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. Date of publication October 2, 2020; date of current version October 27, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB2101300; in part by the Natural Science Foundation of China under Grant 61872147; and in part by the East China Normal University Academic Innovation Promotion Program for Excellent Doctoral Students under Grant YBNLTS2020-041. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue.
(Corresponding author: Mingsong Chen.)

The authors are with the MoE Engineering Research Center of Software/Hardware Co-Design Technology and Application, East China Normal University, Shanghai 200062, China (e-mail: ecnu_hm@163.com; wenxueduan@gmail.com; zhangmin@sei.ecnu.edu.cn; tqwei@cs.ecnu.edu.cn; mschen@sei.ecnu.edu.cn).

Digital Object Identifier 10.1109/TCAD.2020.3012843

0278-0070 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

for performance querying because of its rich programming constructs to model stochastic timing behaviors.

To enable quantitative timing analysis of UML sequence diagrams, this article makes the following major contributions.

1) We extend the syntax and semantics of UML sequence diagrams to support stochastic modeling of system inputs, message processing time, and network delays.
2) We introduce an SMC-based framework with same front-end models but different back-end configurations to enable quantitative analysis of scenario-based designs. A comprehensive set of preprocessing rules is proposed to facilitate the automated conversion from stochastic sequence diagrams into networks of priced timed automata (NPTA) [21].
3) We evaluate the effectiveness of our approach using two industrial designs from the railway domain.

The remainder of this article is organized as follows. After the introduction of related works in Section II, Section III introduces the notations of priced timed automata (PTA). Section IV gives the details of our approach. Section V presents the experimental results. Finally, Section VI concludes this article.

II. Related Work

Scenario-based specifications like UML interaction models [8], [9] and message sequence charts (MSCs) [22] offer an intuitive and visual way of describing design requirements [23]. To ensure the correctness of such specifications, various approaches have been proposed. For example, Alur *et al.* [24] studied the linear temporal logic (LTL) model checking for the class of bounded MSC-graphs. Their work focuses on checking the realizability of bounded MSC-graphs and verification of MSC-graphs. Muram *et al.* [25] presented a model checking-based containment checking approach for UML sequence diagrams, which can be used to verify whether behaviors (or functions) described by a low-level model conform to those specified in its high-level counterpart. Uchitel *et al.* [26] presented a labeled transition system-based method that can describe the closest possible implementation for basic MSCs. Their approach can be used for detecting and providing feedback on the existence of implied scenarios. Although the above methods are promising in detecting functional faults for scenario-based designs, they cannot reason the nonfunctional timing behaviors of target systems.

To enable the modeling of real-time systems, various timing constraints are enforced on scenario-based specifications, such as timing marks [9], timers [22], and interval delays [27]. Consequently, a large spectrum of model checking techniques is developed for the timing analysis of scenario-based specifications. For example, Ju *et al.* [28] presented a schedulability analysis technique for message sequence graph (MSG)-based modeling of distributed real-time systems. By considering the event dependencies specified by MSCs, their approach can be used to produce tight response-time estimates for real-life applications. By reducing the timing analysis problems into linear programming, Li *et al.* [29] conducted reachability analysis, constraint conformance analysis, and bounded delay analysis for UML sequence diagrams and developed a tool named TASS [30] that can analyze UML2.0 interaction models with general and expressive timing constraints for the purpose of timing analysis. By converting time-constrained MSCs into the networks of communicating finite-state machines with local clocks, Akshay *et al.* [31] proposed a model checking approach to verify that all the timed behaviors exhibited by some system conform to the timing constraints imposed by its specification. Although existing model checking-based approaches can guarantee the correctness or real-time performance of systems, most of them focus on safety properties. Few of them support the stochastic behavior analysis of UML sequence diagrams.

SMC [19], [20] has become popular in modeling and quantitative analysis of system performance within an uncertain environment. For example, Du *et al.* [32] adopted the statistical model checker UPPAAL-SMC to evaluate the project schedules with time uncertainties. Chen *et al.* [33] used UPPAAL-SMC to model the thermal and energy variations of MPSoC for the purpose of sustainability evaluation. Basile *et al.* [34] presented their experience in modeling and SMC a satellite-based moving block signaling scenario from the railway industry using UPPAAL-SMC. Although these approaches can be used for the quantitative performance evaluation, most of them focus on specific designs or domains rather than general ones. To support the stochastic behavior modeling of concurrent objects considering the precedence relations between their actions, Gu *et al.* [35] extended UML activity diagrams with timing variations and converts them into NPTA models, which can be used for quantitative performance evaluation. However, this method only considers the stochastic timing behaviors of action executions. It does not take the specific features of sequence diagrams into account.

To the best of our knowledge, our approach is the first attempt that uses SMC for the quantitative timing analysis of UML sequence diagrams considering the variations of system inputs, message processing time, and network delays. Since our approach allows the nested fragments and considers all the fragment notations of sequence diagrams (version UML 2.5), our approach is more powerful and descriptive than traditional MSC-based methods, e.g., MSGs [28] and high-level MSCs (HMSCs) [36]. In other words, MSGs and HMSCs can be easily transformed into their corresponding sequence diagrams with the equivalent semantics. In this way, our quantitative analysis can be applied on the sequence diagram counterparts for the timing performance evaluation performance.

III. Preliminary Knowledge

In our approach, the components of sequence diagrams are encoded as PTA [21], [39], which are a variant of timed automata whose clocks can have different rates in different locations. An NPTA consists of a set of correlated PTAs that communicate with each other using shared variables or broadcast channels. As an example, Fig. 1 shows an NTPA comprising two PTAs, i.e., A $(id = ida)$ and B $(id = idb)$, where each PTA has four locations and two clocks (e.g., Ca and $c1$ for A). We use locations marked with the symbol “C” and “U” to denote the *commit* and *urgent* locations,

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

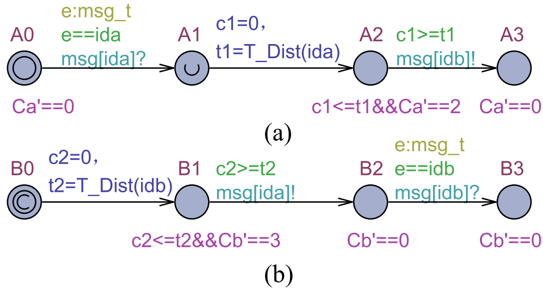


Fig. 1. NPTA, $(A|B)$. (a) PTA A. (b) PTA B.

respectively. Both locations have a delay of 0. In other words, the outgoing transitions of commit and urgent locations must be triggered immediately, whereas the outgoing transitions of commit locations have higher priority.

For each location of a PTA, we can set specific values to primed clocks to denote their rates at that location. If there is no explicit value assignment for a primed clock at some location, the clock has a rate of 1 by default. For example, in location $A0$, we use $Ca' == 0$ to denote that the value of Ca does not change in $A0$. In our approach, we use a message channel array *msg[id]* to perform the synchronization between PTAs, where *id* indicates the target PTA's ID. We adopt *nondeterministic selections* to filter useless broadcasting messages. For example, in PTA A, we use *selections e:msg_t* and guard condition $e == ida$ to monitor incoming messages and filter messages that are not sent to A.

We adopt the pattern as shown in Fig. 1 to model the stochastic timing behaviors of sequence diagrams. Here, the function *T_Dist*() is used to generate time delays following specific distributions. Note that UPPAAL-SMC only supports the uniform and exponential distributions explicitly. However, by proper usage of built-in function random(), we can model a large set of commonly used distributions. For example, based on random(), we can generate the normally distributed random values by using the *Box–Muller* method [40]. Since location $A2$ sets an upper bound for clock $c1$ (i.e., $c1 <= t1$) and its outgoing transition sets a guard condition $c1 >= t1$, PTA A can stay in location $A2$ with an exact duration of $t1$, which follows the distribution indicated by *T_Dist*(*ida*). Based on this pattern, arbitrarily stochastic timing behaviors can be modeled. During SMC, all the randomly simulated runs are dynamically monitored using specified properties in the form of cost-constraint temporal logic. To enable quantitative timing analysis, our approach adopts the properties in the form of $\Pr[<=B](<> \exp)$, where B denotes the time limit, and expression $<> \exp$ checks whether the state predicate *exp* holds eventually. Finally, UPPAAL-SMC will report the success ratio of monitored runs for a given property.

IV. Our Approach

Fig. 2 shows the workflow of our approach. Our approach has two inputs: 1) UML sequence diagrams annotated with uncertainty information (e.g., user inputs, network delays, and message processing time variations) by designers and 2) the design requirements involving the coverage criteria for UML sequence diagrams together with their specific makespan. To facilitate the NPTA model generation, our approach needs to perform the preprocessing for the annotated sequence diagrams, where the fragment constructs, such as *alternatives*, *loops*, and *parallel* are all converted to their corresponding norm forms. Then by using our proposed NPTA model generator, the preprocessed sequence diagrams can be automatically transformed into both front-end and back-end NPTA models. The design requirement information together with the structure information extracted from input sequence diagrams can be used to automate the property-based performance query generation. Based on the statistical model checker UPPAAL-SMC, we can obtain quantitative timing analysis results for the given sequence diagrams. The following sections will detail the major steps of our approach.

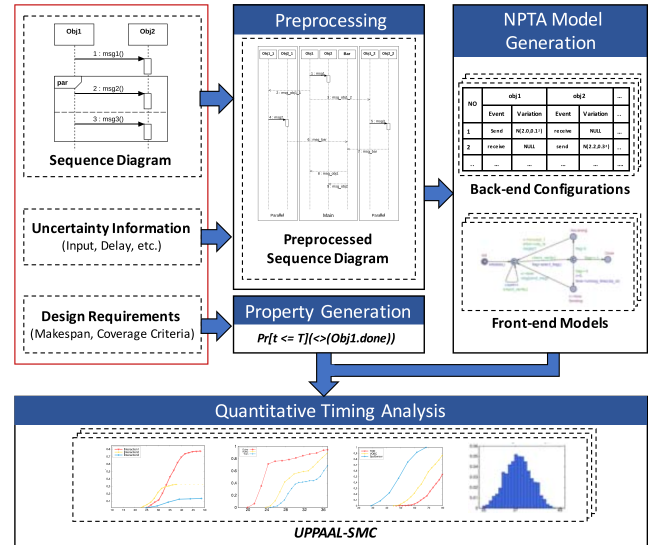


Fig. 2. Workflow of our approach.

A. Modeling of Stochastic Sequence Diagrams

To automate the quantitative timing analysis of UML sequence diagrams, we need to formally and accurately define their stochastic behaviors first. Our approach adopts UML 2.5 [37] as our specification, where sequence diagrams adopt the MSC-like semantics [8]. Since in UML 2.5 [37] sequence diagrams do not support the modeling of uncertainties, we need to extend them by incorporating extra syntax and semantics constructs. Inspired by the basic notations proposed in [38], we define the stochastic sequence diagram as follows.

Definition 1: A *stochastic sequence diagram* is a tuple $\mathrm{SD} = (GV, I, LV, E, L, M, EM, D_m, D_c, F, \mathrm{Exp})$ where:

1) GV is a set of global variables;
2) I is a set of objects with lifelines;
3) $LV = \bigcup_{i \in I} LV_i$ denotes the set of local variables, where LV_i denotes the local variables set for the ith object;
4) $E = \bigcup_{i \in I} E_i$ is a finite set of sending and receiving events, where E_i denotes the set of events on lifeline i;
5) L is a finite set of message labels indicating the functions involving both global and local variables;

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

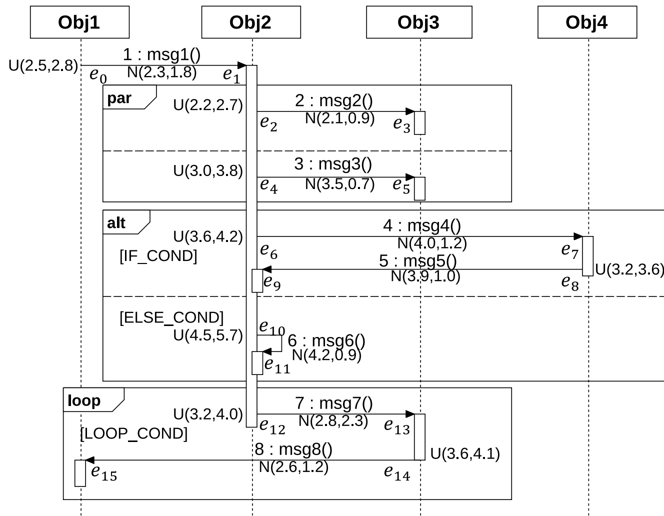


Fig. 3. Example of a stochastic sequence diagram.

6) $M \subseteq E \times L \times E$ is a set of messages, s.t., $(e_{i,1}, l_i, e_{i,2}) \neq (e_{j,1}, l_j, e_{j,2})$ implies $e_{i,1} \neq e_{j,1}$ and $e_{i,2} \neq e_{j,2}$;
7) $EM : E \rightarrow M$ maps an event to its constituting message;
8) $D_m : M \rightarrow DIST$ specifies the distributions of function processing time for every messages;
9) $D_c : M \rightarrow DIST$ specifies the distributions of network delays for every message;
10) F denotes the set of fragments for which the functions *tp*, *ev*, and *nest* are defined as follows.
    a) $tp : F \rightarrow \{par, alt, loop\} \times \mathbb{N}$ specifies the fragment type and the number of operands in the fragment.
    b) $ev : F \times \mathbb{N} \rightarrow 2^E$ denotes the set of events enclosed by an indexed operand of a given fragment.
    c) $nest : F \times \mathbb{N} \rightarrow 2^F$ specifies the set of nested fragments enclosed by an indexed operand of a given parent fragment.
11) $Exp : M \cup F \rightarrow Exp$ is a set of expressions that indicate the guards associated with a message or a fragment.

In UML sequence diagrams, a fragment is a region with a specific meaning according to its type. The current version of sequence diagrams consists of five types of fragments: 1) the *strict* fragment requires that events on a lifeline should occur in the order specified by the partial order P; 2) the *par* fragment has at least two operands where event sequences in different operands can be executed in parallel; 3) the *alt* fragment has at least two operands denoting exclusive alternative event sequences; 4) the *opt* fragment has at least two operands, which enable the *case-switch* operations; and 5) the *loop* fragment has only one operand, which repeats the enclosed event sequences under the specified expression-based guard. In Definition 1, we neglect two kinds of fragments, since *strict* can be encoded using the partial-order relation, and *opt* can be replaced by *alt* without changing the meaning of original sequence diagrams.

Fig. 3 presents an example of a sequence diagram, which consists of four objects, eight messages, and three fragments. For each message in this example, the message processing time follows uniform distribution and network delay follows normal distribution. For example, the processing time of the function *msg2*() associated with message 2 follows the uniform distribution $U(2.2, 2.7)$, which denotes that the minimum execution time is 2.2 ms and its maximum execution time is 2.7 ms. Meanwhile, the network delay of message 2 follows the normal distribution $N(2.1, 0.9)$, which denotes that the mean execution time is 2.1 ms and its standard deviation is 0.9 ms.

We use *partial-order relation* defined in Definition 2 to indicate the visual order [24], which specifies causalities between events within an unrolled stochastic sequence diagram. It is important to note that we treat the occurrences of the same event in different loop iterations as different events.

Definition 2: Let SD' be a stochastic activity diagram, and E be its event set. The *partial-order relation* $P \subseteq E \times E$ denotes the set of happen-before relations between events, s.t., $(e_1, e_2) \in P$ implies e_1 happens before e_2.

We use timed event sequences to indicate dynamic behaviors of UML sequence diagrams.

Definition 3: A *timed event sequence* $\sigma = (e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \cdots \rightarrow (e_m, t_m)$ is a behavior of a stochastic sequence diagram SD', if and only if following conditions hold:

1) all the events listed in σ occur only once;
2) $t_i \leq t_j$ if $i < j$;
3) the execution time of the message $EM(e_i)$ follows the distributions specified by D_m and D_c;
4) the occurrence order of events, i.e., $e_0, e_1, \ldots, e_m$ satisfy the partial order defined by P.

B. Preprocessing of UML Sequence Diagrams

Although previous work [24] uses visual order of events within a sequence diagram to indicate its partial-order relation, most of them adopt the simplified version of sequence diagrams without considering the effects of different types of fragments. Moreover, few of them consider the modeling of implicit *enter* and *exit* events for fragments *par*, *alt*, and *loop*, which signal the fragment boundaries to synchronize their internal events. In this section, we will introduce the preprocessing techniques that conduct proper model transformation for UML sequence diagrams to facilitate specifying the partial-order relation and generating NPTA models. All these preprocessing methods are applied after the parsing of sequence diagrams and the generated preprocessed sequence diagrams are saved as intermediate results within our approach as shown in Fig. 2. Note that since the preprocessing does not modify partial-order relations between events in the original design or incur any timing disturbation for messages, the semantics of the preprocessed UML sequence diagrams remains the same as the original ones. Consequently, the correctness of the following quantitative analysis can be assured.

1) Preprocessing of "par" Fragments: The *par* fragment is used to depict the parallel execution of event sequences within its operands. As an example shown in Fig. 3, the *par* fragment has two operands, and each operand has one message inside. Fig. 4 presents an example to illustrate how our

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

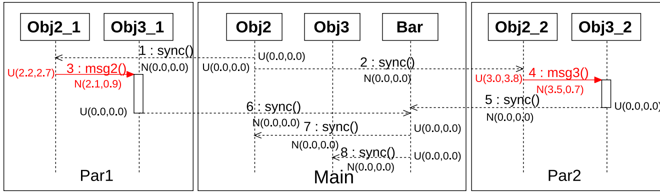


Fig. 4. Preprocessing example of the *par* fragment in Fig. 3.

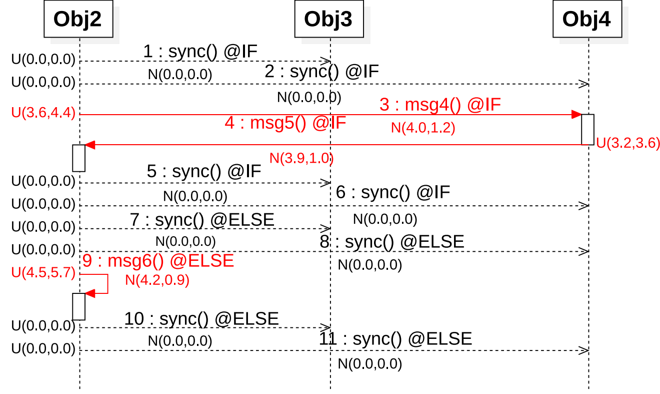


Fig. 5. Preprocessing example of the *alt* fragment in Fig. 3.

approach performs the preprocessing on the fragment. Since the two messages only involve two objects (i.e., *Obj1* and *Obj2*), to enable their parallel executions, our approach duplicates the objects twice, where *Par1* and *Par2* in Fig. 4 show the two copies. Within each copy, we run the corresponding event sequence specified within a *par* fragment operand. Moreover, for each *par* fragment, we introduce one extra object (i.e., *Bar*) to enable fragment-exit synchronization modeling.

To specify the fragment-enter and fragment-exit synchronization operations for a *par* fragment without violating the partial-order relation of its host sequence diagram, our approach introduces multiple auxiliary messages indicated by dashed arrow lines. Note that these auxiliary messages associated with dummy functions *sync*() do not incur any time elapse. [indicated by $N(0.0, 0.0)$]. Their message processing time and network delays are all 0. To model the fragment-enter synchronization, our approach inserts one auxiliary message for each sender of first ready for sending messages in each *par* fragment. For example, we insert a dummy message from *Obj2* to *Obj2_1* with index 1 to trigger the execution of message 3, i.e., *msg2*(). To model the fragment-exit synchronization, the last event receivers in each copy will send the "completion" notifications to the *Bar* object, and the *Bar object* will then notify all the objects involved in the original *par* fragment.

2) Preprocessing of "alt" Fragments: Unlike *par* fragments that support parallel operand execution, *alt* fragments deal with the exclusive operand execution, which does not require the duplication of objects. Fig. 5 illustrates a preprocessing example of the *alt* fragment shown in Fig. 3. During the *alt* fragment preprocessing, we need to mark the branch information on each message within *alt* fragments, which includes both operand indices and the guard expressions of corresponding fragments. For example, in Fig. 3, we use

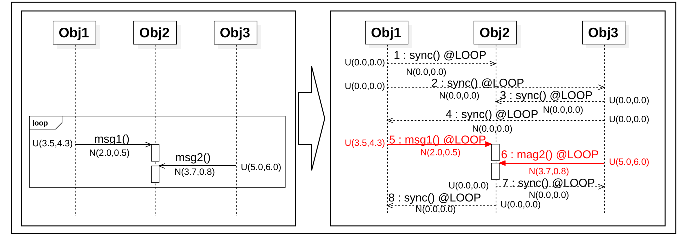


Fig. 6. Preprocessing example of a *loop* fragment.

the notations $msg4()@IF$ and $msg6()@ELSE$ to indicate that $msg4()$ and $msg6()$ are messages in the *if* and *else* branches of the fragment, respectively. To facilitate our NPTA generation, the preprocessing of *alt* fragments needs to figure out the first ready for sending messages in both *if* and *else* branches. For each ready message, assuming its sending object is *obj*, our approach will insert auxiliary messages from *obj* to all the other objects involved in the same *alt* fragment before the execution of the message. As an example in Fig. 5, we insert auxiliary messages 7 and 8 before the execution of message 9, i.e., *msg6*(). To enable the fragment-exit synchronization, assuming that e_i is one of the last receiving events within an *alt* operand, we insert auxiliary messages after $EM(e_i)$ from the host object of e_i to all other objects in the same *alt* fragment. For example, messages 5 and 6 are inserted after message 4 to enable the fragment-exit synchronization for the *if* operand.

3) Preprocessing of "loop" Fragments: Fig. 6 illustrates a preprocessing example of a *loop* fragment. To model the fragment-enter synchronization, our approach needs to figure out the first ready for sending messages and insert auxiliary messages for each of them. As an example shown in case A of Fig. 6, there are two first ready for sending messages (i.e., messages 5 and 6). Similar to the methods used for *alt*, we need to insert two auxiliary messages for each of them. For example, the messages 1 and 2 are inserted for message 5, and messages 3 and 4 are inserted for message 6. To model the fragment-exit synchronization, for objects receiving the last messages (i.e., *Obj2*), we generate auxiliary messages for each of them. As an example shown in case A, messages 7 and 8 are inserted for *Obj2*. In case B, *Obj2* is the only one sender of the first ready for sending message (i.e., message 3), auxiliary messages 1 and 2 are inserted to enable the fragment-enter synchronization. Since there are two receivers for the last messages (i.e., messages 3 and 4), we insert two auxiliary messages for each of them, respectively.

4) Comprehensive Preprocessing Example: Fig. 7 shows how our approach conducts the preprocessing on the stochastic sequence diagram shown in Fig. 3 involving various types of

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

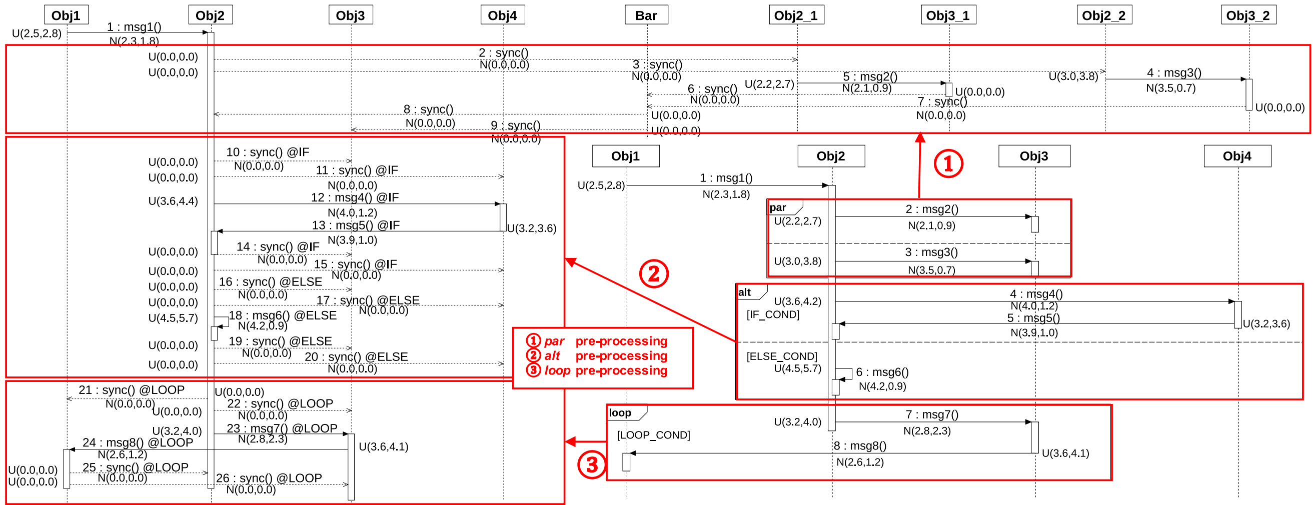


Fig. 7. Preprocessed result of the stochastic sequence diagram shown in Fig. 3.

fragments. By using our proposed methods, we can modularly preprocess the transformation of stochastic sequence diagrams to facilitate the NPTA model generation, while the partial-order relation can be maintained.

C. NPTA Model Generation

Based on the formal definitions in Section IV-A, we can automatically parse stochastic sequence diagrams and figure out the necessary information for the following quantitative timing analysis. Since such extracted information is not good at partial-order relation modeling between events, we need to conduct proper preprocessing on the extracted information to further refine the partial-order relations to facilitate automated generation of NPTA models. To unify NPTA modeling for different stochastic sequence diagrams, our approach adopts both front-end models and back-end configurations to specify their syntax and semantics. In a stochastic sequence diagram, all the objects share the same front-end model and all the messages share another same front-end model. The differences between stochastic sequence diagrams are their distinct back-end configurations, which specify design parameters and design structures to support the execution of sequence diagrams.

1) Front-End Models: We establish one front-end NPTA template for objects with lifelines and messages between objects, respectively. In our approach, we assign each object and message with an ID. In other words, one can identify a specific object or message by a given ID.

The major task of an object instance is to manage a list of events along its lifeline. Table I presents the data structure for an event, where *type* indicates the sending or receiving operation by the event, *channel* is calculated by *encode_msg*() indicating the point-to-point communication between an object and a message (see details in Listing 1), *fragment* calculated by *encode_frag*() encodes the fragment and corresponding operand information for the event. Note that if the event type is *sending*, the object will send a notification to the front-end model of a message rather than an object.

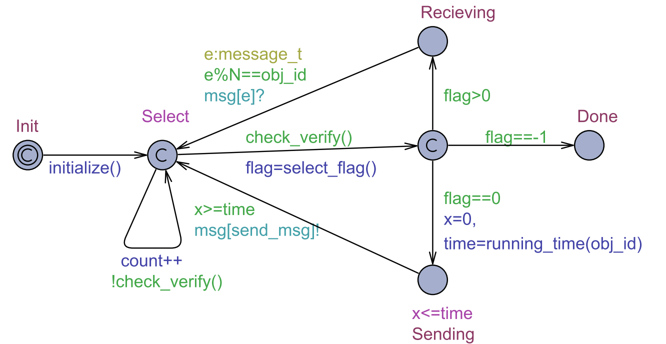


Fig. 8. Front-end model for objects.

Fig. 8 shows the front-end model for an object instance. The following presents its five major locations.

1) *Init* conducts all the necessary initialization for the object instance and all the events that will happen along its lifeline, including the type, broadcast channel, and associated fragment for each event.
2) *Select* determines the execution of current event on the lifeline. The function *check_verify*() tests whether the current event needs to be skipped. If not, the execution of the current event is determined by the flag obtained by calling *select_flag*().
3) *Receiving* indicates that the current object instance will receive a notification from some message instance.
4) *Sending* indicates that the current object instance will send a notification to a message instance via a specific channel with a randomly generated message processing time by *running_time*(*obj_id*).
5) *Done* indicates that lifeline events have been all executed.

Note that after the sending, receiving, or skipping operation, the front-end will go back to the *Select* location to deal with the next event on the lifeline.

In our approach, a message instance tries to receive some notification from its incoming object, execute the associated

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

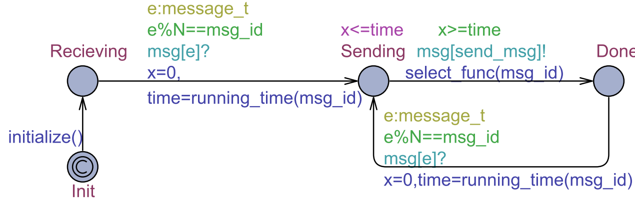


Fig. 9. Front-end model for messages.

TABLE I
ATTRIBUTES OF EVENT DATA STRUCTURE

EVENT_ATTR	Value	Description
type	-1	At the end of lifelines
	0	Sending event
	1	Receiving event
channel	-1	No sending operation
	encode_msg()	Index of Sending Channel
fragment	-1	No host *alt* or *loop* fragments
	encode_frag()	Within some *alt* or *loop* fragment

function, and notify the message completion to the outgoing object. Fig. 9 shows the front-end of a message instance, which consists of four locations as follows.

1) *Init* initializes the message. It figures out the object instances whose lifelines contain the incoming or outgoing events of the message.
2) *Receiving* listens the notification from its incoming object. Once the notification is received, the message instance will immediately switch into *Sending* location.
3) *Sending* indicates the process of message transmission. The duration [calculated by *running_time*(*msg_id*)] spent in this location follows the specified distribution.
4) *Done* denotes the completion of one message instance. If the message is in a *loop* fragment, this message can be repeated again depending on the fragment guard.

2) Back-End Configurations: Back-end models involve both global configuration and local configuration, where the global configuration consists of all the necessary global data structures and functions for the simulation of a stochastic sequence diagram. The local configuration only consists of the necessary local data structures and functions for the simulation of a front-end model. All these configurations can be automatically formed from preprocessed stochastic sequence diagrams.

Listing 1 gives an overview of the global back-end configuration for the stochastic sequence diagram shown in Fig. 3. Here, we use *OBJ*, *PAR_OBJ*, *BAR_OBJ*, *MSG_NUM*, and *FRAG_NUM* to denote the number of objects, the number of duplicated objects for *par* fragments, the number of spawned *BAR* objects for the *par* synchronization, the number of messages, and the number of fragments, respectively. Our approach combines the ID space for both objects and messages. For example, within the NPTA back-end configuration shown in Listing 1, the object ID starts from 0 to 8, while the message ID starts from 9 to 34.

Since UPPAAL-SMC only supports broadcast communication, Listing 1 creates an urgent channel array msg[$N \times N$] to support point-to-point communication. We encode the communication from object/message id_x to message/object

```
1  //global data structures
2  const int OBJ=4,PAR_OBJ=4,BAR_OBJ=1,MSG_NUM=8,FRAG_NUM=2;
3  const int OBJ_NUM=OBJ+PAR+BAR_OBJ;
4  const int N= OBJ_NUM +MSG_NUM;
5  typedef int[0,N] id_t;
6  const int MAX_MSG_OF_OBJ=20;
7  const int EVENT_ATTR =3
8  typedef int[0,OBJ_NUM -1] obj_t;
9  typedef int[OBJ_NUM-1,N-1] msg_t;
10 typedef int[-1,N*N-1] message_t;
11 typedef int[0,FRAG_NUM-1] fg_t;
12 typedef int[0,1] br_t;
13 typedef int[0,1] ftype_t;
14 urgent broadcast chan msg[N*N];
15 double uncertain_input[INPUT_NUM][5],
16        uncertain_msg[N][2][5];
17 ......
18 //Global function definitions
19 void initialize(){......}
20 int encode_msg(id_t sid, id_t rid){return N*sid + rid;}
21 int encode_frag(fg_t fid, br_t bv, ftype_t ft)
22    {return 4*fid + bv*2 + ft;}
23 int conn_graph [OBJ_NUM][MAX_MSG_OF_OBJ][EVENT_ATTR]={
24    {{0,encode_msg(0,9),-1},{1,-1,encode_frag(1,0,1)},
25     {1,-1,encode_frag(1,0,1)},{0,encode_msg(0,1),
26      encode_frag(1,0,1)}, {0,encode_msg(0,2),
27      encode_frag(1,0,1)},{-1,-1,-1}, ...}//obj1 encoding
28     ...... //encoding for other objects}
29 void select_func(msg_t id){
30     if(id==9) msg1(); //call message 1
31     .... // call other messages
32 }
33 double running_time(msg_t id){
34     // return a time following some distribution
35 }
36 ......
37 //Function Library for distributions
38 double normal_dist(double mean, double deviation){...}
39 double Poisson_dist(double lambda){...}
40 ......
```

Listing 1. Back-end configuration for example in Fig. 7.

id_y using the formula $encode_msg(id_x, id_y) = id_x \times N + id_y$, where $msg[encode_msg(id_x, id_y)]$ denotes the private unidirectional channel from id_x to id_y. Let *frag_id*, *frag_type*, *branch* denote the fragment ID, the fragment type (0 means the *alt* fragment, and 1 means the *loop* fragment), and the operand index, respectively. We use the formula $encode_frag(frag_id, branch, frag_type) = frag_id \times 4 + branch \times 2 + frag_type$ to encode the fragment information for an event. To describe the relations between events with different types (see the details from Table I) and object lifelines, we use a 3-D array *conn_graph* to save the events for each object in the order of their occurrences along its lifeline. For example, the first event of *Obj*1 is saved as a tuple $EVT = \{0, encode_msg(0, 9), -1\}$. According to the definition in Table I, the first element of *EVT* indicates that the event type is *sending*. The second element of *EVT* indicates that the front-end model of Obj_1 (ID = 0) will send a notification to the front-end model of message 1 (ID = 9). The third element denotes that the event has no relation with any fragment. In Listing 1, we can also find the functions *select_func*() and *running_time*() which are used by front-end models to execute the message functions and generate a quantity of time following specific distributions, respectively. Note that our global back-end configuration contains a library of distribution functions to support various stochastic behavior modeling. For example, the *initialize*() function can use this library to generate random inputs, message processing time, and network delays

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

```
//local data structures
const int count=0;
int obj[MAX_MSG_OF_OBJ][EVENT_ATTR] =
    conn_graph [obj_id][MAX_MSG_OF_OBJ][EVENT_ATTR];
int maxsize=FRAG_NUM;
......
//Local function definitions
bool check_verify(){
    if(obj[count][2]==-1) return true; // no fragment
      ......
    //discuss the case of alt fragment
    //discuss the case of loop fragment
}
int select_flag(){
    //obtain the event type information from conn_graph
}
......
```

Listing 2. Back-end configuration for an object.

based on the distribution parameters saved in data structures *uncertain_input* and *uncertain_msg*.

Listing 2 presents the back-end configuration for an object. In our approach, each object has a local variable *count* which indicates the index of the currently checking event along the lifelines. The function *check_verify*() is used to check whether the current event is executable. For example, according to Table I, $obj[count][2] == -1$ indicates that the current event has no relation with any fragment. Therefore, it can be safely executed. Otherwise, if the event belongs to an *alt* fragment, *check_verify*() will check the guards of both of its *if* and *else* operands. If the guard does not hold, the enclosing event will be skipped. Otherwise, the *check_verify*() will return *true* to allow the event execution. The things become more complex when dealing with *loop* fragments. In the global *initialization*() function, all the events within a loop will record the position of the first message in the loop. When the last message finishes, the function *check_verify*() will check the guard of the loop. If the guard holds, the variable *count* will be assigned with the position of the first message in the same loop. Otherwise, *check_verify*() will continue the following messages after the loop. In the back-end configuration, we use the function *select_flag*() to obtain the event type information from the data structure *conn_graph*.

D. Property Generation and Quantitative Analysis

As shown in Fig. 2, the design requirement for the property generation contains the information including the required makespan of the evaluation targets and the coverage criteria for sequence diagrams. Since there exist various uncertainties during the interaction with the external world, it is hard to fully guarantee that a specified functional scenario can be finished for a given time limit. Instead, designers would like to know "what is the probability that a functional scenario can be accomplished within a time limit?".

Based on the specified coverage criteria, our approach can automatically extract the functional scenarios from the given sequence diagrams. In the current version of our approach, we mainly consider three kinds of coverage criteria as follows: 1) *object coverage* requires that all the object completion (i.e., the object has handled all its lifeline events) should be covered; 2) *interaction coverage* denotes that all the interactions between messages should be covered; and 3) *subscenario coverage* indicates that all the possible branches caused by *alt* fragments should be covered. Since our approach is based on the model checker UPPAAL-SMC, the generated properties are in the form of $\Pr[<=T](<> expr)$. For a given NPTA model generated by some sequence diagram, such properties can be used to check the probability that the functional scenarios described by *expr* can complete within time limit T. Typically, for functional scenarios extracted by the above coverage criteria, our approach adopts the following query templates.

1) *Object queries* are in the form of $\Pr[<=T] <> (Obj_i.Done)$, which checks the probability that Obj_i can complete within time limit T.
2) *Interaction queries* are in the form of $\Pr[<=T](<> (Event_i.Status \&\& Event_j.Status))$, which denotes the probability of causal relations or overlapped executions between events within time limit T.
3) *Subscenario queries* are in the form of $\Pr[<=T](<> (Event_0.Done \&\& \ldots \&\& Event_n.Done))$, which indicates the probability that all the subscenario indicated by the completion of events (i.e., $Event_0, \ldots, Event_n$) can finish within time limit T.

Based on the parameters ϵ (probability uncertainty) and α (probability of false negatives) set by designers, UPPAAL-SMC can figure out the number of runs to produce an approximation interval $[p-\epsilon, p+\epsilon]$ with a confidence of $1-\alpha$. After the execution of the random runs, the distribution of the probability of successful simulations will be used for quantitative analysis.

V. Performance Evaluation

To evaluate the effectiveness of our approach, we implemented our framework using the JAVA programming language. By parsing the XMI files of extended sequence diagrams drawn by the UML edit tool Enterprise Architect, our tool can automatically generate both NPTA models and property-based performance queries to enable the SMC. In the experiment, we used UPPAAL-SMC (version 4.1.18 with $\epsilon = 0.05$ and $\alpha = 0.05$) as the engine of our framework. All the experiments were performed on a desktop with 3.40-GHz Intel Core i5 CPU and 16-GB RAM.

In this experiment, we collected two case studies in the railway domain from our industrial partner Casco Signal Ltd., including: 1) an operating mechanism for restrict manual forward (RMF) mode switching and 2) a control system for overspeed supervision and protection (OSP). Both designs are modeled using the UML sequence diagrams. We collected the uncertainties (i.e., system inputs, message processing time, and network delays) about these two scenarios from railway signal experts of Casco, and specified such information in corresponding UML sequence diagrams. To enable automated evaluation, performance queries are automatically generated from the provided coverage criteria (i.e., object coverage, subscenario coverage, and interaction coverage). The following two sections show the details of our experiments.

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

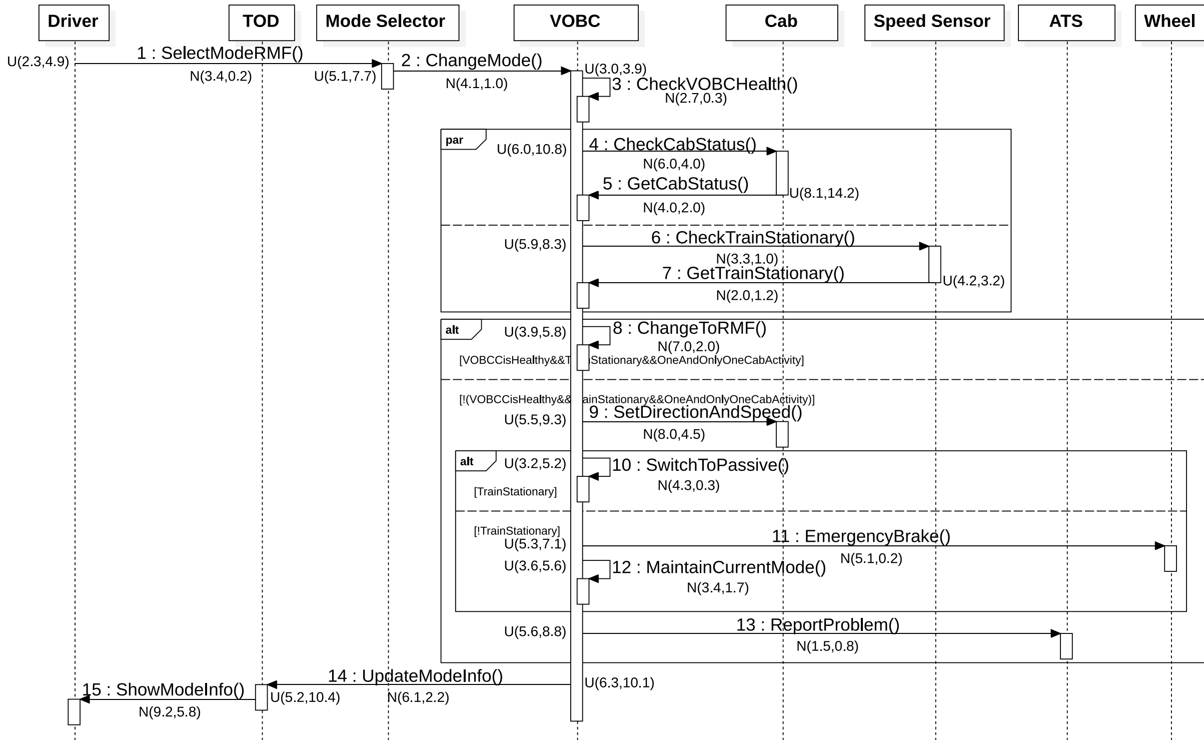


Fig. 10. Sequence diagram for RMF mode switching.

A. Experiment 1—RMF Mode Switching

Fig. 10 presents the UML sequence diagram describing the scenario of RMF mode switching. This sequence diagram was extracted from the design manual of Casco. According to the design manual, after the request of mode switching indicated by messages 1 and 2, the vehicle onboard controller (VOBC) will check the states of the cab and the train indicated by message 3 and the *par* fragment. Once the self-inspection is done, the VOBC will determine which mode will be selected by using the nested *alt* fragments. In this example, there are three choices: 1) if the outer fragment guard is satisfied, the VOBC will switch into RMF mode; 2) if the outer fragment guard is not satisfied but the inner fragment guard is satisfied, the VOBC will switch into passive mode; and 3) if none of the outer and inner fragment guards is satisfied, the VOBC will apply emergency braking on wheels (see message 11) and do not change its current state (see message 12), meanwhile a problem report will be generated and sent to the automatic train supervision (ATS) component. The UML sequence diagram consists of eight objects and 15 messages. Based on the statistical data (collected from emulation traces) provided by Casco experts, we figured out the message processing and network delays in normal distributions as shown in Fig. 10. We applied all the built-in coverage criteria in this experiment, Due to the space limitation, we only present typical performance queries for each coverage category.

By using object queries, we can investigate the probabilities of object completion for a given time limit. In this experiment, we set the time limit to 80 ms. Fig. 11 shows the SMC results for three object queries, i.e., Pr [<=80](<> *VOBC.Done*), Pr [<=80](<> *TOD.Done*), and Pr [<=80](<> *SpdSensor.Done*)). Note that the x-axis and y-axis indicate the completion time and success ratio of desired functional scenarios specified by corresponding properties. The three queries quantitatively evaluate the completion performance of objects VOBC, train operator display (TOD), and speed sensor (SS) under the time limit, respectively. By simulating 199 runs, 400 runs, and 36 runs, the three queries can achieve probability intervals [0.81, 0.91], [0.49, 0.59], and [0.90, 1] with a confidence of 95%, respectively. The SMC checking for each query needs more than 7 h. From this figure, we can find that the object SS can achieve the highest performance, since it only involves two events with in the second operand of the *par* fragment.

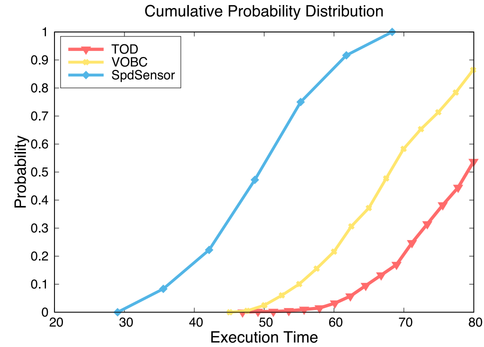


Fig. 11. Object queries for RMF mode switching.

The RMF mode switching consists of a *par* fragment, which involves the concurrent execution of multiple messages. To investigate the causal relation and overlap between concurrent messages, we conducted the interaction queries.

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

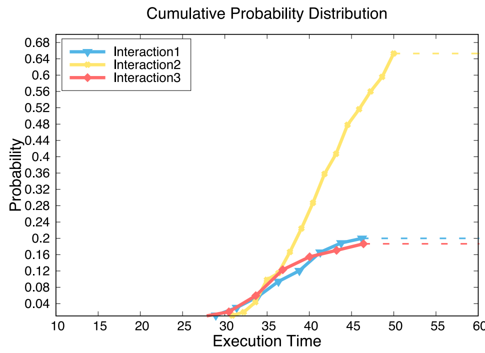


Fig. 12. Interaction queries for RMF mode switching.

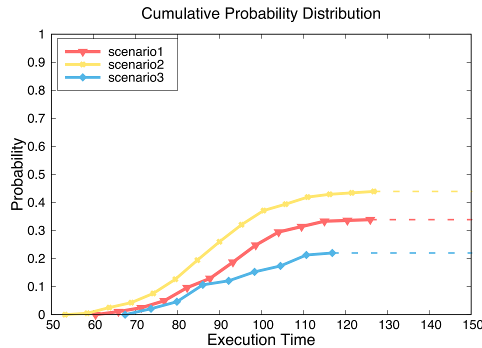


Fig. 13. Subscenario queries for RMF mode switching.

Fig. 12 presents three interaction scenarios using the following properties.

1) *Interaction 1:* Pr [<=50](<> $m5$.*Done*&&$m7$.*Sending*) that checks the probability that message $m5$ is received while message $m7$ is in transmission within 50 ms.
2) *Interaction 2:* Pr [<=50](<> $m5$.*Sending*&&$m7$.*Done*) that checks the probability that message $m7$ is received while message $m5$ is in transmission within 50 ms.
3) *Interaction 3:* Pr [<=50](<> $m5$.*Sending*&&$m7$.*Sending*) that checks the probability that both messages $m5$ and $m7$ are in transmission simultaneously within 50 ms.

The evaluation for each of these three properties requires around 4 h. From this figure, we can find that the occurrence chance of interaction 2 where $m7$ finishes before the sending of $m5$ is much higher than occurrence chances of the other two interactions, since the combination of messages 4 and 5 requires more message processing time and message sending time than the combination of messages 6 and 7.

Due to the nested *alt* fragments, Fig. 13 contains three subscenarios. We conducted the subscenario queries using the following three properties.

1) *Scenario 1:* Pr [<=200](<>$m1$.*Done*&&$m8$.*Done* && $m14$.*Done* &&$m15$.*Done*) which checks the completion probability of a subscenario comprising messages $m1$, $m8$, $m14$, and $m15$ within 200 ms.
2) *Scenario 2:* Pr [<=200](<>$m1$.*Done*&&$m10$.*Done* && $m14$.*Done* &&$m15$.*Done*) which checks the completion probability of a subscenario comprising messages $m1$, $m10$, $m28$, and $m29$ within 200 ms.

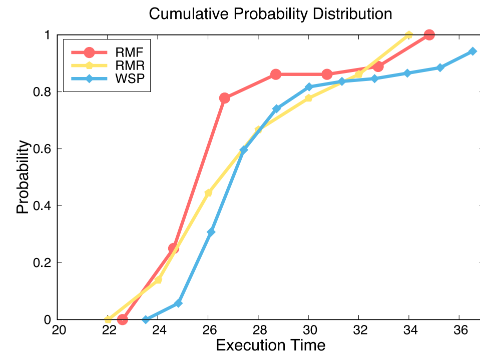


Fig. 14. Object queries for OSP w/ different drive modes.

3) *Scenario 3:* Pr [<=200](<>$m1$.*Done*&&$m11$.*Done* && $m14$.*Done* &&$m15$.*Done*) which checks the completion probability of a subscenario comprising messages $m1$, $m11$, $m14$, and $m15$ within 200 ms.

The evaluation of each of these three queries needs more than 20 h. From this figure, we can observe that the VOBC has a high chance to switch into the passive mode (i.e., the probability interval is $[0.39, 0.49]$ with a confidence of 95%), and the chance to apply emergency braking is low (i.e., the probability interval is $[0.17, 0.27]$ with a confidence of 95%).

B. Experiment 2—Overspeed Supervision and Protection

In the second experiment, we investigated a UML sequence diagram describing the OSP mechanism supported by VOBCs. The OSP sequence diagram consists of seven objects ad 12 messages. It has one *par* fragment (containing three messages), and one *alt* fragment (containing four messages). All the message processing time and network delays follow normal distributions, and the input for the five drive modes follows the uniform distribution.

The sequence diagram describes a scenario where an onboard train controller communicates with a train control center to ensure the safety of trains. At the beginning, the train driver should choose a drive mode (RMF, RMR, WM, OFF, and WSP) which allows a different maximum speed. During the driving, the onboard controller continuously monitors the speed of its host train. When a train approaches its permitted speed, the VOBC controller will send an alert to TOD. Once the driver notices the alert, he/she will take proper actions to slow down the train speed. For example, if the train speed exceeds the limit for a specific drive mode, the VOBC controller will report the overspeed information to TOD and apply the emergency braking immediately to slow down the train. It is important to note that the network delay and message execution variations caused by hardware and software components play an important role in determining the accuracy of the control, which strongly influence the real-time performance of the train. In this example, we focus on the queries for the object VOBC, which is the key part of the train control system.

First, we evaluated the impacts of drive modes on the object completion time by fixing the value for the drive mode input. Fig. 14 presents the evaluation results for VOBC in different drive modes with a time limit (i.e., 37 ms). The object queries

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

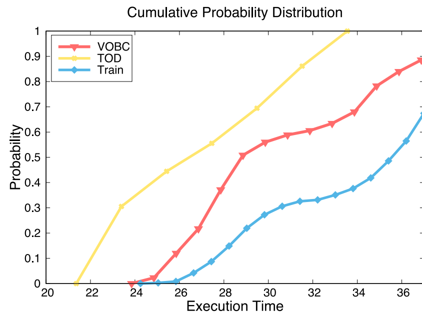


Fig. 15. Object queries for OSP.

are in the form of $\Pr[<=37](\mathit{VOBC.Done}\&\&\mathit{mode} = x)$ where $x \in \{\mathit{RMF}, \mathit{RMR}, \mathit{WSP}\}$. By executing 36 runs, the first property ($x = \mathit{RMF}$) and the second property ($x = \mathit{RMR}$) achieve a probability interval $[0.90, 1]$ with a confidence of 95%, respectively. By executing 104 runs, the third property ($x = \mathit{WSP}$) obtains a probability interval $[0.88, 0.98]$ with a confidence of 95%. All these three queries cost more than 10 h to obtain the evaluation results. From this figure, we find that although the events of VOBC can be completed within 37 ms for all three modes, the drive mode RMF can achieve the best performance, since it has a higher chance to achieve the shortest completion time when the execution time is smaller than 33 ms. Meanwhile, we can find that the drive mode RMF has the worse completion time, since its average completion time is the largest among the three modes.

We also investigated the case where the drive mode inputs are randomly selected based on a uniform distribution, i.e., each mode can have a 20% chance to be selected. We conducted the performance queries on all the objects, where Fig. 15 only presents three out of them. The queries for the listed ones are in the form of $\Pr[<=37](<> \mathit{VOBC.Done})$, $\Pr[<=37](<> \mathit{TOD.Done})$, and $\Pr[<=37](<> \mathit{Train.Done})$. By executing 175 runs, 175 runs, and 352 runs, the properties can achieve probability intervals of $[0.83, 0.93]$, $[0.90, 1]$, and $[0.62, 0.72]$ with a confidence of 95%, respectively. All these three queries cost more than 12 h to obtain the evaluation results. From this figure, we can find that TOD can be finished within the given time limit and VOBC has a better completion time than the train object.

VI. Conclusion and Future Work

Along with the prosperity of CPSs, more and more software systems need to interact with external uncertain environment frequently. However, due to the lack of modeling and evaluation mechanisms, traditional scenario-based requirement analysis methods cannot be used to capture and quantify the stochastic timing behaviors of key system scenarios. Although there exist multiple approaches for the timing analysis of UML sequence diagrams, most of them adopt the simplified timing constraints and focus on the functional correctness or the reachability analysis. None of them can be used for the quantitative timing analysis considering various uncertainties. To address this issue, this article extends the semantics of UML sequence diagrams and proposes an effective framework that can automatically conduct quantitative analysis of UML sequence diagrams by translating them into formal NPTAs. The experimental results on two industrial case studies from the railway domain demonstrate the efficacy of our approach.

Note that although SMC requires much less verification effort than traditional model checking methods, it is still very time consuming to conduct performance queries on complex scenarios of large designs. For example, in experiment 1 (i.e., RMF model switching), the SMC-based performance evaluation of each subscenario query needs more than 7 h, which is not acceptable in practice. Therefore, how to reduce the checking time is an interesting topic that is worthy of future study.

References

[1] E. A. Lee, "Cyber-physical systems: Design and challenges," in *Proc. Object Orient. Real Time Distrib. Comput. (ISORC)*, 2008, pp. 363–369.
[2] S. A. Seshia, S. Hu, W. Li, and Q. Zhu, "Design automation of cyber-physical systems: Challenges, advances, and opportunities," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1421–1434, Sep. 2017.
[3] C. Tsigkanos, T. Kehrer, and C. Ghezzi, "Modeling and verification of evolving cyber-physical spaces," in *Proc. ACM SIGSOFT Conf. Found. Softw. Eng. (FSE)*, 2017, pp. 38–48.
[4] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, "Uncertainty-wise cyber-physical system test modeling," *Softw. Syst. Model.*, vol. 18, no. 2, pp. 1379–1418, 2019.
[5] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, 2012, pp. 573–583.
[6] M. Chechik, R. Salay, T. Viger, S. Kokaly, and M. Rahimi, "Software assurance in an uncertain world," in *Proc. Int. Conf. Fundam. Approaches Softw. Eng. (FASE)*, 2019, pp. 3–21.
[7] M. Famelis and M. Chechik, "Managing design-time uncertainty," *Softw. Syst. Model.*, vol. 18, no. 2, pp. 1249–1284, 2019.
[8] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2005.
[9] J. Rumbaugh and I. Jacobson, *The Unified Modeling Language Reference Manual*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2004.
[10] S. Dziwok, C. Gerking, S. Becker, S. Thiele, C. Heinzemann, and U. Pohlmann, "A tool suite for the model-driven software engineering of cyber-physical systems," in *Proc. ACM SIGSOFT Conf. Found. Softw. Eng. (FSE)*, 2014, pp. 715–718.
[11] S. Abrahão, C. Gravino, E. Insfran, G. Scanniello, and G. Tortora, "Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: Results from a family of five experiments," *IEEE Trans. Softw. Eng.*, vol. 39, no. 3, pp. 327–342, Mar. 2013.
[12] C. Ghezzi and A. M. Sharifloo, "Quantitative verification of non-functional requirement with uncertainty," in *Dependable Computer Systems*. Heidelberg, Germany: Springer, 2011, pp. 47–62.
[13] M. Pan and X. Li, "Timing analysis of MSC specifications with asynchronous concatenation," *Int. J. Softw. Tools Technol. Transfer*, vol. 14, no. 6, pp. 639–651, 2012.
[14] M. Ahmad, C. Cnaho, J. M. Bruel, and R. Laleau, "How to handle environmental uncertainty in goal-based requirements engineering," in *Proc. Int. Conf. Softw. Eng. (ICSE) Companion*, 2018, pp. 368–369.
[15] E. Letier, D. Stefan, and E. T. Barr, "Uncertainty, risk, and information value in software requirements and architecture," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, 2014, pp. 883–894.
[16] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, 2011, pp. 585–591.
[17] C. Ghezzi and A. M. Sharifloo, "Model-based verification of quantitative non-functional properties for software product lines," *Inf. Softw. Technol.*, vol. 55, no. 3, pp. 508–524, 2013.
[18] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, 2004, pp. 202–215.
[19] K. G. Larsen, "Priced timed automata and statistical model checking," in *Proc. Int. Conf. Integr. Formal Methods (IFM)*, 2013, pp. 154–161.

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.

[20] A. David, K. G. Larsen, M. Mikucionis, and D. B. Poulsen, "Uppaal SMC tutorial," *Int. J. Softw. Tools Technol. Transfer (STTT)*, vol. 17, no. 4, pp. 397–415, 2015.

[21] A. David *et al.*, "Statistical model checking for networks of priced timed automata," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst. (FORMATS)*, 2011, pp. 80–96.

[22] "Message sequence chart," Int. Telecommun. Union Stand. Sector, Geneva, Switzerland, Recommendation Z.120, 2011.

[23] A. Sutcliffe, "Scenario-based requirements analysis," *Requirements Eng.*, vol. 3, no. 1, pp. 48–65, 1998.

[24] R. Alur, K. Etessami, and M. Yannakakis, "Realizability and verification of MSC graphs," *Theor. Comput. Sci.*, vol. 331, no. 1, pp. 97–114, 2005.

[25] F. U. Muram, H. Tran, and U. Zdun, "A model checking based approach for containment checking of UML sequence diagrams," in *Proc. Asia Pac. Softw. Eng. Conf. (APSEC)*, 2015, pp. 73–80.

[26] S. Uchitel, J. Kramer, and J. Magee, "Detecting implied scenarios in message sequence chart specifications," in *Proc. Int. Symp. Found. Softw. Eng. (FSE)*, 2001, pp. 74–82.

[27] H. Ben-Abdallah and S. Leue, "Timing constraints in message sequence chart specifications," in *Proc. Int. Conf. Formal Techn. Netw. Distrib. Syst. (FORTE)*, 1997, pp. 91–106.

[28] L. Ju, A. Roychoudhury, and S. Chakraborty, "Schedulability analysis of MSC-based system models," in *Proc. IEEE Real Time Embedded Technol. Appl. Symp. (RTAS)*, 2008, pp. 215–224.

[29] X. Li, M. Pan, L. Bu, L. Wang, and J. Zhao, "Timing analysis of scenario-based specifications using linear programming," *Softw. Test. Verification Rel.*, vol. 22, no. 2, pp. 121–143, 2012.

[30] M. Pan, L. Bu, and X. Li, "TASS: Timing analyzer of scenario-based specifications," in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, 2009, pp. 689–695.

[31] S. Akshay, P. Gastin, M. Mukund, and K. N. Kumar, "Model checking time-constrained scenario-based specifications," in *Proc. Found. Softw. Technol. Theor. Comput. Sci. (FSTTCS)*, 2010, pp. 204–215.

[32] D. Du, M. Chen, X. Liu, and Y. Yang, "A novel quantitative evaluation approach for software project schedules using statistical model checking," in *Proc. Int. Conf. Softw. Eng. (ICSE) Companion*, 2014, pp. 476–479.

[33] M. Chen, X. Zhang, H. Gu, T. Wei, and Q. Zhu, "Sustainability-oriented evaluation and optimization for MPSoC task allocation and scheduling under thermal and energy variations," *IEEE Trans. Sustain. Comput. (TSUSC)*, vol. 3, no. 2, pp. 84–97, Apr.–Jun. 2018.

[34] D. Basile, M. H. Beek, and V. Ciancia, "Statistical model checking of a moving block railway signalling scenario with Uppaal SMC—Experience and outlook," in *Proc. Int. Symp. Leverag. Appl. Formal Methods Verification Validation (ISoLA)*, 2018, pp. 372–391.

[35] F. Gu, X. Zhang, M. Chen, D. Große, and R. Drechsler, "Quantitative timing analysis of UML activity diagrams using statistical model checking," in *Proc. Design Autom. Test Europe (DATE)*, 2016, pp. 780–785.

[36] T. H. Kim and S. D. Cha, "Timed high-level message sequence charts for real-time system design," in *Proc. Int. Workshop Syst. Anal. Model. (SAM)*, 2006, pp. 82–98.

[37] "OMG unified modeling language TM (OMG UML), superstructure version 2.5," Object Management Group, Needham, MA, USA, Rep. formal/2015-03-01, 2015.

[38] S. Sieverding, C. Ellen, and P. Battram, "Sequence diagram test case specification and virtual integration analysis using timed-Arc Petri nets," in *Proc. Int. Workshop Formal Eng. Approaches Softw. Compon. Archit. (FESCA)*, 2013, pp. 17–31.

[39] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and Z. Wang, "Time for statistical model checking of real-time systems," in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, 2004, pp. 349–355.

[40] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Ann. Math. Stat.*, vol. 29, no. 2, pp. 610–611, 1958.

Ming Hu (Member, IEEE) received the B.E. degree from the School of Computer Science and Software Engineering, East China Normal University, Shanghai, China, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Embedded Software and System.

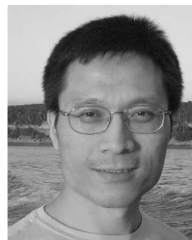
His research interests include the area of program analysis, design automation of cyber-physical systems, and software testing.

Wenxue Duan received the B.E. degree from Shanghai Polytechnic University, Shanghai, China, in 2017, and the M.E. degree from the Software Engineering Institute, East China Normal University, Shanghai, in 2020.

Her research interests include the area of design automation of cyber-physical systems and software engineering.

Min Zhang (Member, IEEE) received the B.S. degree in computer science from Shandong Normal University, Jinan, China, in 2005, the M.S. degree in software theory from Shanghai Jiao Tong University, Shanghai, China, in 2008, and the Ph.D. degree in software science from the Japan Advanced Institute of Science and Technology (JAIST), Nomi, Japan, in 2011.

From 2011 to 2014, he was a Postdoctoral Researcher with JAIST. He is currently an Associate Professor with the Software Engineering Institute, East China Normal University, Shanghai. His research interests include formal methods, programming languages, and software engineering, particularly on embedded systems and intelligent systems.



Tongquan Wei (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from Michigan Technological University, Houghton, MI, USA, in 2009.

He is currently an Associate Professor with the School of Computer Science and Technology, East China Normal University, Shanghai, China. His research interests are in the areas of green and reliable embedded computing, cyber-physical systems, parallel and distributed systems, and cloud computing.

Dr. Wei has been serving as a Regional Editor for the *Journal of Circuits, Systems, and Computers* since 2012. He also served as a Guest Editor for several special sections of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS and *ACM Transactions on Embedded Computing Systems*.

Mingsong Chen (Senior Member, IEEE) received the B.S. and M.E. degrees from the 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, FL, USA, in 2010.

He is currently a Professor with the Software Engineering Institute, East China Normal University, Shanghai, China. His research interests are in the area of embedded system, design automation of cyber-physical systems, parallel and distributed systems, and formal verification techniques.

Prof. Chen is an Associate Editor of *IET Computers & Digital Techniques* and the *Journal of Circuits, Systems and Computers*.

Authorized licensed use limited to: East China Normal University. Downloaded on November 03,2020 at 00:59:44 UTC from IEEE Xplore. Restrictions apply.