2009 IEEE/ACM International Conference on Automated Software Engineering

A Formal Syntax for Probabilistic Timed Property Sequence Charts

Pengcheng Zhang†, Lars Grunske*, Antony Tang*, Bixin Li†
†*School of Computer Science and Engineering, Southeast University, Nanjing, China*
E-mail: {pchzhang,bx.li}@seu.edu.cn
**Faculty of ICT, Swinburne University of Technology, Hawthorn, VIC 3122, Australia*
E-mail: {lgrunske,atang}@swin.edu.au

***Abstract*—Probabilistic properties are considered as the most important requirements for a variety of software systems, since they are used to formulate extra-functional requirements such as reliability, availability, safety, security and performance requirements. Currently, several probabilistic logics have been proposed to specify such important properties. However, due to the inherent complexity of the underlying temporal logics, these probabilistic logics are rather complex and software developers have problems using them to correctly specify the intended properties. To overcome this problem, we define a formal and graphical property specification language called Probabilistic Timed Property Sequence Charts (PTPSC) which is a probabilistic extension of Property Sequence Charts (PSC). We illustrate the use of PTPSC in the context of a vehicle-to-vehicle communication device for avoiding traffic accidents.**

***Keywords*-Probabilistic Properties, Property Sequence Chart, Probabilistic and Timed Property Sequence Chart.**

I. Introduction

Probabilistic properties are considered as the most important requirements for software in among other medical, avionic, automotive and telecommunication systems [13]. These probabilistic properties are required to formulate extra-functional requirements such as reliability, availability, safety, security and performance requirements.

To specify probabilistic properties, probabilistic temporal logics such as PCTL (Probabilistic Computation Tree Logic) [9], PCTL* [4], PTCTL (Probabilistic Timed CTL) [11] and CSL (Continuous Stochastic Logic) [5] have been proposed. Although for these logics, probabilistic specification patterns [7] enriched with a structured English grammar have been proposed, the textual notations are rather complex and software developers have problems using them to correctly specify the intended properties. To ease the specification, a graphical specification formalism for probabilistic properties is desired. This specification formalism needs to balance *expressive power* and *simplicity of use*, i.e., the specification should be as simple as possible, without losing expressive power.

Based on these two design rationales Autili et al. [3] have already proposed a non-probabilistic scenarios-based property specification notation called Property Sequence Chart (PSC). PSC provides a complete graphical front-end for software developers so they do not have to deal with any particular textual or logical formalism. In our previous work [16], PSC has been enriched with time constructs (called Timed PSC or TPSC) to specify timing properties for real-time systems. However, current PSC and TPSC still cannot specify probabilistic properties. Consequently, in order to help software developers to specify probabilistic properties we propose a probabilistic extension of PSC and TPSC, called PTPSC.

The rest of the paper is organized as follows: Section II and III introduce an example and provide some background on PSC and TPSC. Section IV describes the main contributions of this paper, the PTPSC language. Section V concludes the paper and presents a list of future work.

II. An Illustrative Example

To illustrate the concepts described in this paper we use a system for preventing car collisions as an example. This system uses Dedicated Short Range Communication (DSRC) devices for vehicle-to-vehicle (V2V) communication to notify neighboring vehicles of their relative positions [15]. Vehicles that are equipped with the device can communicate their positions and compute collision trajectories [12]. DSRC communication goes through two steps: (i) establishing communication through handshaking, and each vehicle would assign a channel for communicating data; (ii) communicate positional data, i.e. GPS, at regular intervals (one second), when the channels are free. If the two vehicles are close enough and a collision might occur, the V2V device would warn the driver. The warning should appear at a point when there is enough time for the driver to react and avoid an accident. However, if the latency of communication is high, say one second, and the channels are busy or the vehicle position is not communicated in time, it is possible that by the time the vehicle position is sent through to a neighboring vehicle, the collision is imminent. In a study [10], it has been found that the probability of DSRC message reception is 82% at 0m distance, and drops off to 30% at 300m distance, assuming there are no objects such as other vehicles obstructing the line of sight. An improved DSRC device developed in Australia has recorded improved reliability of 90% at 100m distance without any obstructions. We are using this data in our model.

Let us consider a probabilistic scenario between two vehicles. Assuming the two vehicles head in the same direction, one behind another with a distance of 100m, the probability that a message can be exchanged successfully every second is 90%. If the forward vehicle *V1* is stationary and the trailing vehicle *V2* is traveling at 100km/hr, there

1527-1366/09 $29.00 © 2009 IEEE
DOI 10.1109/ASE.2009.56

is 7.3s before *V2* must stop to avoid colliding with *V1*, it would depend on the V2V device receiving the message in time to warn the driver and the driver in *V2* acting upon that warning immediately – with a probability of 0.99 [15]. This real-life example comprises both the probabilistic and timing properties that affect the reliability and safety requirements of the collision avoidance system. We use PTPSC to model a simple scenario involving two vehicles, and it can be expanded to model scenarios involving multiple vehicles.

III. Preliminaries: PSC and TPSC

PSC is an extended graphical notation of a subset of UML 2.0 sequence diagrams, which is proposed in [3] to specify temporal properties. Figure 1 shows the PSC graphical elements. A PSC graphical specification is composed of a set of component instances, messages, constraints and operators. Two basic message types are available: *arrowMSGs* and *intraMSGs*. The *arrowMSGs* have three subtypes: *Regular*, *Required* and *Fail*. Regular messages (labeled with *e:msg*) are used to define the precondition for a desired (or an undesired) interaction. Required messages (labeled with *r:msg*) must be exchanged by the system and are used to express mandatory interactions. Fail messages (labeled with *f:msg*) should never be exchanged and are used to express undesired interactions. *IntraMSGs* are used to describe *constraints* that restrict the future and past exchange of messages (*arrowMSGs*). *Constraints* are classified into two categories: *unwanted message constraints*, *chain constraints*. An *unwanted message constraint* is specified for a set of *intraMSGs* the system must not exchange. In other words, an unwanted message constraint describes the event(s) or interactions that are disallowed between two component instances. *Chain constraints* are defined as a sequence of dependent *intraMSGs*, and are further classified as *wanted* and *unwanted*. *Wanted chain constraints* are satisfied if the messages are exchanged following the sequence imposed by the chain specifications. *Unwanted chain constraints* require that the messages do not occur in the sequence specified in the chain specification.

Constraints are also classified into *past constraints*, and *future constraints*. *Past constraints* specify message exchanges, wanted or unwanted, before a specific message exchange event takes place, and *future constraints* specify the constraints afterwards. Graphically, past constraints are closely located to the arrow source and future constraints are closely located to the arrow target of an *arrowMSGs*. Formally, *arrowMSGs* and the different constraints types can be defined as follows:

Definition 1: **(*ArrowMSG* Constraints)** Let C, AM, and IM denote the finite set of component instances, *arrowMSGs* and *intraMSGs* in the system. Let $\underline{m}$ be a message label, b be an unwanted message constraint, and g be a chain constraint, which are formally defined as follows:

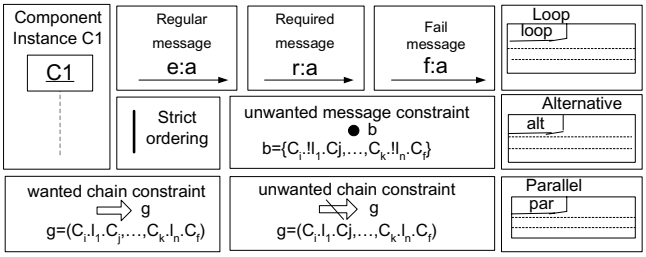

Figure 1. The PSC Graphical Notation

- $\underline{m} = c_i.m.c_j$, where $m \in AM \cup IM, c_i, c_j \in C$, which denotes a message exchanged between two components;
- b=$\{\bar{m_1}, \bar{m_2}, ..., \bar{m_n}\}$, where $\bar{m_i} = c_{j_i}.!m_i.c_{k_i}$ $(i = 1, ..., n)$, $m_i \in IM$; $c_{j_i}, c_{k_i} \in C$, and "$!m_i$" means m_i is not exchanged;
- g=$(\underline{m}_1, \underline{m}_2, ..., \underline{m}_n)$, where $\underline{m}_i = c_{j_i}.m_i.c_{k_i}$ $(i = 1, ..., n)$, $m_i \in IM$; $c_{j_i}, c_{k_i} \in C$.

PSC has five operators: *loose*, *strict*, *parallel*, *loop* and *alt*, which define how *arrowMSGs* can be composed. The *loose* operator defines the order of messages, however any other messages can occur between the messages. The *strict* operator explicitly specifies a strict ordering between a pair of messages; no other message is allowed in between. The *parallel*, *loop* and *alt* operators specify parallel merging (i.e., interleaving), iteration and alternative behavior, respectively.

In our previous work [16], we have proposed a timed extension of PSC, called Timed PSC or TPSC, based on annotated clock constraints and clock reset.

Definition 2: **(Clock Constraints)** For a set of clocks X, a clock constraint δ from the set of clock constraints $\Phi(X)$ can be defined as follows [1], [2]:

$\delta := x < c \mid x \leq c \mid x > c \mid x \geq c \mid \neg\delta \mid \delta_1 \wedge \delta_2$

where $x \in X$ is a clock variable, and $c \in \mathbb{N}$ is a constant, assuming discrete time.

Two functions $\models$ and $[[\delta]]$ need to be defined. The function $\models$ evaluates for each value of a clock $v(x)$ if the clock constraint is fulfilled or not. The function $[[\delta]]$=$\{v(x) \mid v(x) \models \delta\}$ denotes all the values of a clock which satisfy δ. We assume that a clock constraint δ is homogenous, meaning that the clock constraint is fulfilled only for a single connected set of clock values. Two additional functions are defined: $[[pre(\delta)]]$=$\{x_1 \mid x_1 < min([[\delta(x)]])\}$ and $[[succ(\delta)]]$=$\{x_2 \mid x_2 > max([[\delta(x)]])\}$ that describe all clock values that do not fulfill the clock constraint and which happen before and after the clock constraint.

Example ▷ For example, if $\delta(x) = 2 \leq x \wedge x \leq 4$, $[[pre(\delta)]] = \{v(x) \mid v(x) < 2\}$, and $[[succ(\delta(x))]] = \{v(x) \mid v(x) > 4\}$. ◁

Definition 3: **(Clock Reset)** A clock reset for a clock x is defined as $v(x) := 0$. Normally, we use ψ as a set of clock resets.

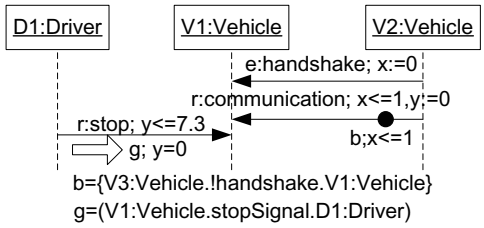

Figure 2. A TPSC property example

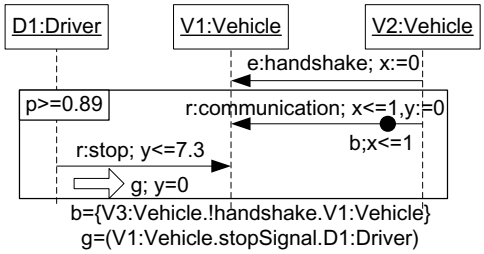

Figure 3. A PTPSC property example

Based on these definitions, the PSC *arrowMSGs* and the different constraint types as given in Definition 1 can be extended with *clock constraints* and *clock reset*. We refer to [16] for a detailed description of the TPSC syntax and semantics.

Example ▷ Figure 2 shows a TPSC specification of a scenario in the given example without probability. It shows that *V1* may receive a *handshake*(a *regular* message) from *V2* and the two vehicles must have communicated a *required* message with each other. There is a *past unwanted message* constraint b which means that there are no other vehicles, such as *V3* in the example, to *handshake* with *V1*. Let's assume that *V1* is travelling in the same direction as *V2* and the velocity of *V1* is 100km/h, where *V2* has stalled and stopped on the road around the bend, and that they are 100m apart. *V2* broadcasts its location to *V1*, there is 7.3s elapsed time for the driver of *V1* to *stop* (a required message) and avoid the collision [15]. Before this message, there is a *past wanted chain* constraint g which means *V1* must send *stopSignal* to the driver first. While the TPSC specification is enough to represent timing properties for a real-time system, it cannot be used to represent probabilistic properties as in our engineering example. ◁

IV. Probabilistic Timed Property Sequence Charts (PTPSC)

This section defines an extension of the PSC [3] and TPSC [16] property specification formalisms, called PTPSC. First the informal ideas of using probability in TPSC are explained. Then a precise and structured syntax definition of PTPSC is followed.

A. Extending TPSC with Probability

The idea of adding probability into TPSC is motivated by the work of Refsdal et al. [14], which adds probability to UML sequence diagrams. In this work each message or operator can be annotated with a probability. However, there are some slight differences between PSC and UML sequence diagrams. Firstly, TPSC is a specification for timing properties. Secondly, the messages in TPSC represent different types, so the semantics of adding probability constructs to these different types of messages is also different. Thirdly, according to the idea of live sequence charts [6], we add pre-charts to PTPSC. The messages in the pre-charts are restricted to the *regular* type. Following the pre-chart in PTPSC, a main-chart is enriched with a probability operator.

Example ▷ Figure 3 shows the PTPSC property of the example. According to the scenario, the probability for message *communication* is 0.9 and for message *stop* is 0.99. Consequently, the joint probability for these two independent messages to happen is $0.99 * 0.9 \approx 0.89$. As explained before, we can add a set of messages with a probabilistic operator, so the two messages *r:communication* and *r:stop* are added with a probabilistic operator of ≥ 0.89. ◁

B. PTPSC Syntax

The PTPSC syntax is given in Table I. An *italic* setting is used for non-terminals. Literal terminals are delimited by quotation marks (""). We use the terminal symbol $\bullet$ to denote the sequential composition of two messages and the notation $\downarrow$ to connect two operands in an operator. The symbol "λ" is used to denote the empty constraint of an *arrowMSG*. The symbols δ, δ', and δ'' are used to represent clock constraints for messages, past constraints and future constraints, respectively, and they are defined in Definition 2. ψ, ψ', and ψ'' represent the sets of reset clocks for messages, past constraints and future constraints, respectively. They are defined in Definition 3. *ArrowMSG* constraints ($PastCon$ and $FutureCon$) which can be $UnwantedMsg$, $WantedChain$ or $UnwantedChain$ constraints are defined in Definition 1. $MsgLabel$ is a message label which is also defined in Definition 1. Rule (1) shows the main structure of PTPSC which is composed of a pre-chart ($PreTPSC$) and a main-chart ($TPSC \bullet (Op \mid Op_{Fail})$) with a probability $p_{req}(0 \leq p_{req} \leq 1)$, where $\bowtie \in \{<, \leq, >, \geq\}$.

The rules of the grammar are divided into three categories: rules for the pre-chart (2-15), rules for the main-charts (16-35) and rules for the messages (36-43). The rules for the pre-chart can be further grouped into rules for the pre-chart operators $PreOp$ (3-10) and rules for the message sequences ($PreMsgSeq$) (11-15) that are used as operands by the operator rules. In a pre-chart only regular messages (labeled with *e:msg*) are allowed. As defined in Rule (2)

Start	(1) $PTPSC$	:==	$PreTPSC + \mathcal{P}_{\bowtie p_{req}}(TPSC \bullet (Op \mid Op_{Fail}))$
$PreTPSC$	(2) $PreTPSC$	:==	$\varepsilon \mid PreTPSC \bullet PreOp$
$PreOp$	(3) $PreOp$	:==	$PreLoose \mid PreStrict \mid PrePar \mid PreAlt \mid PreLoop$
	(4) $PreLoose$	:==	"Loose("+$PreMsgSeq_{NoCon} \mid PreMsgSeq_{Past}$+")"
	(5) $PreStrict$	:==	"Strict("+$PreMsgSeq_{AllNoCon}$+")"
	(6) $PrePar$	:==	"Par("+$Operand_{PrePar}$+")"
	(7) $PreLoop$	:==	"Loop("+$PreMsgSeq_{NoCon} \mid PreMsgSeq_{Past}$+")"
	(8) $PreAlt$	:==	"Alt("+$Operand_{PreAlt}$+")"
	(9) $Operand_{PrePar}$	:==	$(\varepsilon \mid Operand_{PrePar}) \downarrow PreMsgSeq_{AllNoCon}$
	(10) $Operand_{PreAlt}$	:==	$(\varepsilon \mid Operand_{PreAlt}) \downarrow (PreMsgSeq_{NoCon} \mid PreMsgSeq_{Past})$
$PreMsgSeq$	(11) $PreMsgSeq$	:==	$PreMsgSeq_{NoCon} \mid PreMsgSeq_{Past} \mid PreMsgSeq_{Future}$
	(12) $PreMsgSeq_{NoCon}$	:==	$(\varepsilon \mid PreMsgSeq) \bullet eMsg_{NoCon}$
	(13) $PreMsgSeq_{Past}$	:==	$(\varepsilon \mid PreMsgSeq_{Past} \mid PreMsgSeq_{NoCon}) \bullet eMsg_{Past}$
	(14) $PreMsgSeq_{Future}$	:==	$(\varepsilon \mid PreMsgSeq) \bullet eMsg_{Future}$
	(15) $PreMsgSeq_{AllNoCon}$	:==	$(\varepsilon \mid PreMsgSeq_{AllNoCon}) \bullet eMsg_{NoCon}$
$TPSC$	(16) $TPSC$	:==	$\varepsilon \mid TPSC \bullet Op$
Op	(17) Op	:==	$Loose \mid Strict \mid Par \mid Alt \mid Loop$
	(18) $Loose$	:==	"Loose("+ $MsgSeq_{Past} \mid MsgSeq_{NoCon}$+")"
	(19) $Strict$	:==	"Strict("+$MsgSeq_{AllNoCon}$+")"
	(20) Par	:==	"Par("+$Operand_{Par}$+")"
	(21) $Loop$	:==	"Loop("+$MsgSeq_{Past} \mid MsgSeq_{NoCon}$+")"
	(22) Alt	:==	"Alt("+$Operand_{Alt}$+")"
	(23) $Operand_{Par}$	:==	$(\varepsilon \mid Operand_{Par}) \downarrow MsgSeq_{AllNoCon}$
	(24) $Operand_{Alt}$	:==	$(\varepsilon \mid Operand_{Alt}) \downarrow (MsgSeq_{Past} \mid MsgSeq_{NoCon})$
Op_{Fail}	(25) Op_{Fail}	:==	$Loose_{Fail} \mid Strict_{Fail} \mid Alt_{Fail}$
	(26) $Loose_{Fail}$	:==	"Loose("+$MsgSeq_{Fail}$+")"
	(27) $Strict_{Fail}$	:==	"Strict("+$MsgSeq_{NoCon} \bullet fMsg_{NoCon}$ +")"
	(28) Alt_{Fail}	:==	"Alt("+$Operand_{Fail}$+")"
	(29) $Operand_{Fail}$	:==	$(\varepsilon \mid Operand_{Alt} \mid Operand_{Fail}) \downarrow MsgSeq_{Fail}$
$MsgSeq$	(30) $MsgSeq$	:==	$MsgSeq_{Future} \mid MsgSeq_{Past} \mid MsgSeq_{NoCon}$
	(31) $MsgSeq_{Past}$	:==	$(\varepsilon \mid MsgSeq_{Past} \mid MsgSeq_{NoCon}) \bullet (eMsg_{Past} \mid rMsg_{Past})$
	(32) $MsgSeq_{Future}$	:==	$(\varepsilon \mid MsgSeq) \bullet (eMsg_{Future} \mid rMsg_{Future})$
	(33) $MsgSeq_{NoCon}$	:==	$(\varepsilon \mid MsgSeq) \bullet (eMsg_{NoCon} \mid rMsg_{NoCon})$
	(34) $MsgSeq_{AllNoCon}$	:==	$(\varepsilon \mid MsgSeq_{AllNoCon}) \bullet Msg_{NoCon}$
	(35) $MsgSeq_{Fail}$	:==	$((\varepsilon \mid MsgSeq_{Past} \mid MsgSeq_{NoCon}) \bullet fMsg_{Past}) \mid ((\varepsilon \mid MsgSeq) \bullet fMsg_{NoCon})$
$ArrowMsg$	(36) $eMsg_{NoCon}$	:==	"e:"+$MsgLabel$+";"+δ+","+ψ+"λ"
	(37) $eMsg_{Past}$	:==	"e:"+$MsgLabel$+";"+δ+","+ψ+$PastCon$+δ'+","+ψ'
	(38) $eMsg_{Future}$	:==	"e:"+$MsgLabel$+";"+δ+","+ψ+$FutureCon$+";"+δ''+","+ψ''
	(39) $rMsg_{NoCon}$	:==	"r:"+$MsgLabel$+";"+δ+","+ψ+"λ"
	(40) $rMsg_{Past}$	:==	"r:"+$MsgLabel$+";"+δ+","+ψ+$PastCon$+δ'+","+ψ'
	(41) $rMsg_{Future}$	:==	"r:"+$MsgLabel$+";"+δ+","+ψ+$FutureCon$+";"+δ''+","+ψ''
	(42) $fMsg_{NoCon}$	:==	"f:"+$MsgLabel$+";"+δ+","+ψ+"λ"
	(43) $fMsg_{Past}$	:==	"f:"+$MsgLabel$+";"+δ+","+ψ+$PastCon$+δ'+","+ψ'

Table I
PTPSC SYNTAX

a pre-chart can be empty (denoted by ε) or a sequence of operators $PreOp$ in the pre-chart. If the last message of a sequence has a future constraint and the first message of the next sequence has a past constraint, constraint conflict may occur. Consequently, we restrict that the last message of an operand in *Loose*, *Alt* and *Loop* only have past constraints or no constraints, as given by $PreMsgSeq_{Past}$ and $PreMsgSeq_{NoCon}$. To provide a clear semantics, the operators *Par* and *Strict* can contain only messages without any constraints. Consequently, a special type of message sequence ($PreMsgSeq_{AllNoCon}$) is defined for these operators. With Rule (13) we disallow past constraints on messages if the previous message has a future constraint.

The rules for the main-chart can be further grouped into rules for the operators Op (17-24), rules for the operators Op_{Fail} (25-29) where the last message is a *fail* message, and rules for the message sequences ($MsgSeq$) (30-35). Op and $MsgSeq$ are defined in the similar way in pre-chart except that *required* messages are also permitted. Op_{Fail} operators contain message sequences, which end with a

fail message $MsgSeq_{Fail}$. Since no other messages should follow a fail message, operators Par and $Loop$ cannot have fail messages.

The syntax of basic *arrowMSGs* are defined in Rules (36-45). A fail message cannot have a future constraint because when a fail message happens the future of the system does not need to be considered [3].

Example ▷ Parsing from right to left, the property of the example can be defined by the derivation sequence (1,17,18, 31,40,31[ε],40,16[ε],2,3,4,12[ε],36,2[ε]). Most of the rules describe the composition of the messages, whereas Rule (36) defines the message *e:handshake*, and Rule (40) is used for the messages *r:communication* and *r:stop*. ◁

V. Conclusion and Future Work

In this paper we have defined a formal syntax for a probabilistic property specification language called PTPSC. Based on the PTPSC syntax, PTPSC can be used to help designers to specify probabilistic properties for real-time system. We have also shown how to use PTPSC in a case study for specifying a probabilistic property of an automotive safety device to avoid traffic accidents.

In the future, we plan to define a full formal semantics for the proposed PTPSC specification based on the semantic domain Timed Büchi Automaton (TBA) and statistical hypothesis testing. We will also investigate how to integrate PTPSC in different verification environments, such as model checking and run-time verification. In [8], an approach has been proposed to monitor probabilistic properties specified by a subset of CSL. The approach may also be modified and used for PTPSC specification.

Acknowledgment

This work is supported partially by the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2007513, partially by National High Technology Research and Development Program under Grant No.2008AA01Z113, partially by the National Natural Science Foundation of China under Grant No.60773105 and partially by the Program for New Century Excellent Talents in University under Grant No.NCET-06-0466.

Correspond to Bixin Li, bx.li@seu.edu.cn

References

[1] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and Computation*, vol. 104, no. 1, pp. 2–34, 1993.

[2] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[3] M. Autili, P. Inverardi, and P. Pelliccione, "Graphical scenarios for specifying temporal properties: an automated approach," *Automated Software Engineering*, vol. 14, no. 3, pp. 293–340, 2007.

[4] A. Aziz, V. Singhal, and F. Balarin, "It usually works: The temporal logic of stochastic systems," in *Proc. 7th Int. Conference on Computer Aided Verification, CAV 95*, ser. LNCS, vol. 939. Springer, 1995, pp. 155–165.

[5] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time markov chains," in *Proc. 10th International Conference on Concurrency Theory, CONCUR 99*, ser. LNCS, J. C. M. Baeten and S. Mauw, Eds., vol. 1664. Springer, 1999, pp. 146–161.

[6] W. Damm and D. Harel, "LSCs: Breathing life into message sequence charts," *Formal Methods in System Design*, vol. 19, no. 1, pp. 45–80, 2001.

[7] L. Grunske, "Specification patterns for probabilistic quality properties," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, Robby, Ed. ACM, 2008, pp. 31–40.

[8] L. Grunske and P. Zhang, "Monitoring probabilistic properties," in *Proc. 7th joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) ESEC-FSE 09*, 2009, pp. 183–192.

[9] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[10] D. Jiang, V. Taliwal, A. Meier, W. Holfelder, and R. Herrtwich, "Design of 5.9 GHz DSRC-based vehicular safety communication," *IEEE Wireless Communication*, vol. 16, no. 2, pp. 36–43, 2006.

[11] M. Z. Kwiatkowska, G. Norman, D. Parker, and J. Sproston, "Performance analysis of probabilistic timed automata using digital clocks," *Formal Methods in System Design*, vol. 29, no. 1, pp. 33–78, 2006.

[12] U. D. of Transportation, *Vehicle Safety Communications Project*. Washington D.C.: National Highway Traffic Safety Administration, 2006.

[13] A. Pretschner, M. Broy, I. H. Krüger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *International Conference on Software Engineering, ICSE 2007, Workshop on the Future of Software Engineering, FOSE 2007*, L. C. Briand and A. L. Wolf, Eds., 2007, pp. 55–71.

[14] A. Refsdal, K. E. Husa, and K. Stølen, "Specification and refinement of soft real-time requirements using sequence diagrams," in *FORMATS '05*, ser. LNCS, vol. 3829. Springer, 2005, pp. 32–48.

[15] A. Tang and A. Yip, "Collision avoidance timing analysis of DSRC-based vehicles," *Accident Analysis and Prevention*, p. In Press, 2009.

[16] P. Zhang, B. Li, and M. Sun, "A timed extension of property sequence chart," in *11th IEEE High Assurance Systems Engineering Symposium(HASE'08)*. Nanjing, China: IEEE Computer Society, 2008, pp. 197–206.