Comput Sci Res Dev manuscript No.
(will be inserted by the editor)

From Software Verification to 'Everyware' Verification

Marta Kwiatkowska

Received: date / Accepted: date / Preliminary version

Abstract Ubiquitous computing is a vision of computing in which the computer disappears from view and becomes embedded in our environment, in the equipment we use, in our clothes, and even in our body. Everyday objects – called 'everyware' by Adam Greenfield – are now endowed with sensing, controlled by software, and often wirelessly connected and Internet-enabled. Our increasing dependence on ubiquitous computing creates an urgent need for modelling and verification technologies to support the design process, and hence improve the reliability and reduce production costs. At the same time, the challenges posed by ubiquitous computing are unique, deriving from the need to consider coordination of communities of 'everyware' and control physical processes such as drug delivery.

Model-based design and verification techniques have proved useful in supporting the design process by detecting and correcting flaws in a number of ubiquitous computing applications, but are limited by poor scalability, efficiency and range of scenarios that can be handled. In this paper we describe the objectives and initial progress of the research aimed at extending the capabilities of quantitative, probabilistic verification to challenging ubiquitous computing scenarios. The focus is on a advancing quantitative verification in new and previously unexplored directions, including game-based techniques, incorporation of continuous dynamics in addition to stochasticity, and online approaches. The research involves investigating the fundamentals of quantitative verification, development of algorithms and prototype implementations, and experimenting with case studies.

Keywords Ubiquitous computing · Probabilistic, real-time and hybrid models · Quantitative specifications · Quantitative/Probabilistic model checking · Runtime verification

1 Introduction

In the words of Adam Greenfield, "the age of ubiquitous computing is here: a computing without computers, where information processing has diffused into everyday life, and virtually disappeared from view" [52]. Ubiquitous computing, also known as pervasive computing, was conceived by Mark Weiser over 20 years ago in a paper [100] which famously began as follows: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it". In the ubiquitous computing world, a multitude of sensor-enabled computing devices, of all shapes and sizes, are operating, silently supporting our daily activities and autonomously making decisions on our behalf. These software-controlled devices are embedded in our environment, in the equipment we use, in our clothes, and even in our body. As information processing becomes part of the functionality of everyday objects, these objects evolve from conventional hardware and software into 'everyware' [52], which refers to artefacts that incorporate sensing, computation and communication, and are 'smart', that is, capable of sensing what is around them, remembering the context and adapting to new situations, and communicating with humans as well as other devices. When wirelessly connected and Internet-enabled, these smart everyday objects form the 'Internet of Things', envisaged as early as 1999 [94] and organised into communities, networks and ecosystems. The fast pace of techno-

Supported in part by ERC Advanced Grant VERIWARE and Institute for the Future of Computing, Oxford Martin School.

M. Kwiatkowska
Department of Computer Science, University of Oxford, Oxford OX1 3QD, UK
E-mail: Marta.Kwiatkowska@cs.ox.ac.uk

logical progress in electronics, communication engineering and cloud computing has turned this powerful vision into reality, with remotely managed home appliance networks and Internet-enabled fridges [1] available on the market today.

The widespread adoption of the ubiquitous computing paradigm has been facilitated by a number of factors, the main one being the huge growth in the mobile phone and tablet platform, which now outstrips the desktop PC, particularly in the third world countries. Thanks to the advances in microelectronics, smartphones are now endowed with several miniature sensors, e.g. accelerometers and GPS, and can be used to perform a wide range of information gathering functions, including air pollution monitoring, location-sensitive look up, and even more specialised healthcare checks such as ECG monitoring [33]. Sensor-enabled smart devices are also used to monitor and control physical processes, for example, self-parking and self-driving cars, environmental and wildlife monitoring, as well as implantable devices such as glucose monitors and cardiac pacemakers [93, 62]. Future potential developments in this area are endless, with nanotechnology and molecular sensing devices already envisaged [67].

Our growing dependence on sensor-enabled smart devices, together with a steep increase in the number of device recalls [3], have naturally prompted a surge of interest in methodologies for ensuring their safety, reliability and performability. At the core of these devices is embedded software, for which a variety of design and validation methodologies exist. Model-based design and automated verification technologies offer a number of advantages, particularly with regard to embedded software controllers: they enable rigorous software engineering methods such as model checking in addition to testing, and have the potential to reduce the development effort through code generation and software reuse via product lines. Models can be extracted from high-level design notations or even source code, represented as finite-state abstractions, and systematically analysed to establish if, e.g., the executions never violate a given property. These technologies provide means to automatically analyse properties such as "the smartphone will never execute a financial transaction more than once" (reliability), "the probability of failure to raise alarm if the levels of airborne pollutant are unacceptably high is tolerably low" (safety), and "the maximum expected time to retrieve location information based on GPS position is within the specified range" (performability).

Automated verification via model checking [32] has made great progress in recent years, resulting in a variety of software tools now integrated within software development environments, to mention CProver [29]. These have been applied to limited ubiquitous computing scenarios, for example, sensor network software for TinyOS [13]. In cases where the focus is on safety, reliability and performance, it is necessary to include in the models quantitative aspects such as probability, time delays and energy usage. The preferred technique here is quantitative, probabilistic verification [68], which employs variants of Markov chains, annotated with real-time and costs/rewards, as models and aims to establish quantitative properties, for example, calculating the real-time deadline, probability of an event of interest, or expected cumulated cost of some process. Tools such as the real-time model checker UPPAAL [8] and the probabilistic model checker PRISM [72] are widely used for this purpose in several application domains, including distributed systems, communication networks and wireless protocols.

However, the world of ubiquitous computing poses new and wide-ranging challenges. The UK Ubiquitous Computing Grand Challenge [18] put forward a three-pronged proposal, pertaining to the design and engineering of 'everyware' systems, their scientific understanding, and human interaction mechanisms; see also [78]. Robin Milner was a great proponent of developing the science of ubiquitous computing [85], believing that "ubiquitous computing can empower us, if we can understand it". Within the extensive landscape of research into the science of ubiquitous computing, the work reported here, inspired by Milner's vision, offers a focused contribution that complements his foundational research with a more practical, algorithmic slant towards industrially relevant methodologies and software tools to support the design of ubiquitous computing devices. More specifically, we aim at extending quantitative verification techniques towards considering the following aspects: communities of 'everyware', which act autonomously, can exhibit cooperative and competitive behaviour due to conflicting goals, potentially giving some agents an unfair advantage that can lead to unexpected gains/losses; sensor-enabled devices frequently need to monitor and control physical processes, such as the electrical signal in the heart or concentration of glucose in the blood, which necessitates the consideration of sophisticated, possibly non-linear continuous dynamics in addition to discrete function and stochasticity; and continuous interaction with the environment naturally leads to adaptive behaviours, where offline verification no longer suffices and online verification is essential to ensure dependability of reconfigurations. As concrete examples of situations that we are targeting, consider the following properties where the new aspects are highlighted in italics: "the expected energy usage will remain within the specified range, *irrespective of demand by other users*" (competitive); "the blood glucose level will return to a normal range in at most 3 hours, *assuming wireless communication failure rate is within specified tolerance*" (continuous and stochastic dynamics); and "the expected time to make a collective decision by a group of potentially faulty mobile sensors falls within a specified interval of time, *even allowing for some sensors to move out of reach of the signal*".

This paper reports on ongoing research performed as part of the ERC Advanced Grant VERIWARE (2010-15) [4], aimed at developing automated quantitative verification and synthesis methodologies to cater for ubiquitous computing. We summarise the state of the art and the goals of the VERIWARE project, and present selected research highlights from the initial phase: (i) analysis of cooperative behaviours [26]; (ii) integration of sensing and monitoring for continuous physical processes, including at nanoscale [24, 79]; (iii) compositional quantitative verification via multi-objective probabilistic model checking [73,46,47], and (iv) quantitative runtime verification to handle adaptive behaviours [14,48]. We conclude by outlining the challenges and future promise of the methods.

2 State of the Art

Quantitative modelling. Model-based design methodologies support the design process through providing tools for a range of analysis methods, as well as code generation. Ubiquitous computing systems are distributed and reactive, and therefore naturally modelled using automata-like formalisms or process calculi, where the focus is on system dynamics as it evolves. Mobility and spatial movement characteristic of ubiquitous computing systems can be modelled in e.g. the pi-calculus [86]. The induced models are state-transition graphs, where vertices correspond to system states and arrows correspond to transitions between system states. Quantitative extensions annotate model states or transitions with real-valued quantities that capture resource usage. These include weighted automata [20], which can model a variety of resource types, as well as probabilistic [98], timed [6] and hybrid automata [58], which target specific resources and associated analyses. Probabilistic extensions of process calculi have also been proposed, for example the probabilistic pi-calculus [88], where transitions are taken according to discrete probability distributions, and the stochastic process algebra PEPA [60], where transitions are governed by exponential distributions.

The majority of the modelling formalisms for representing reactive systems have discrete dynamics, where transitions between the states are discrete. Quantitative extensions of such formalisms may redefine this dynamics in a number of ways: as discrete transitions but with a probabilistic outcome, as in probabilistic automata [98]; as continuous real-valued time delay, as in timed automata [6]; or as continuous stochastically distributed time delay, as in stochastic process algebras [60]. All these features – and additional information about resources, e.g. energy, memory consumption, etc – play a part in the modelling of 'everyware', whose underlying semantics is *hybrid*. By virtue of being embedded [59], 'everyware' are exposed to reaction constraints, which include time deadlines and which can be hard or stochastically distributed, as well as execution constraints, such as energy level and failures in the (wireless) communication medium. The design of such systems must combine control engineering techniques (for the reaction constraints, essentially achieved through deriving analytical models) with computational techniques (for the execution constraints, achieved via extended automata-based models). Despite recent progress concerning design frameworks and tool support for embedded systems, e.g. Ptolemy [40], the integration of discrete and continuous dynamics, particularly in presence of stochastic behaviour, remains a challenge for the long term.

Automated verification via model checking. Model checking [32,11] is a formal verification technique that can establish automatically, that is, by means of an algorithm, whether certain properties, usually expressed in temporal logic, hold for a system model, typically a finite-state automaton. Temporal logics such as CTL view the executions as a tree and can express properties (of states or paths) which refer to the relative order of events, for example, whether the target state is reachable from the initial state. More complex path properties, such as repeated reachability or fairness, can be expressed in LTL which views the system dynamics as a set of executions. CTL model checking proceeds inductively on the structure of the formula, with the temporal subformulas established through a fixpoint computation. The LTL model checking procedure transforms the negation of the formula into a finite-state automaton and then checks the reachability on the product of thus obtained automaton and the system model. For both CTL and LTL graph-theoretic algorithms suffice for automatic verification. The model checking technology is well established, with several model checking tools widely available and in industrial use. The main limitation of model checking is the state-space explosion problem, which can be tackled by techniques such as model reduction, compositional reasoning and abstraction techniques.

Quantitative verification. Quantitative model checking approaches have emerged in three main variants: *real-time* and *hybrid* system verification, respectively based on timed and hybrid automata, and the more recent *probabilistic verification*, the core of this project. Timed automata are automata endowed with clocks, and allow for time delays specified as dense real-time values. The underlying semantics gives rise to infinite-state dynamics, which can be reduced to appropriate finite-state representations by the region or zone-based quotients. Efficient model checking algorithms exist on the zone quotient, implemented in the UPPAAL model checker. Hybrid automata allow continuous flows according to differential equations, for example to model thermodynamics, and the corresponding model checking algorithms, in view of undecidability, are restricted to certain subclasses, with implementation in PHAVer [49]. Both UP-

PAAL and PHAVer have been applied to the analysis of embedded systems that are relevant to ubiquitous computing, such as automotive controllers and mobile robots, but do not support stochastic dynamics. Probabilistic verification (also known as *probabilistic model checking*) is founded on [98] and subsequently developed for further model classes [68]. It applies to probabilistic models, typically variants of Markov chains, and has been implemented in the tools PRISM [72], PROBMELA [9] and MRMC [64]. Model checking for probabilistic extensions of process algebras can be achieved through translation to Markov chains [87]. There are a variety of models, which can be classified according to whether they contain discrete or continuous state spaces or probability distributions, exhibit nondeterminism, or feature continuous time or variables. The models can be endowed with costs and rewards that annotate states and/or transitions. Similarly to timed and hybrid automata, abstraction techniques can be applied to yield finite-state models amenable to model checking.

Quantitative specifications are stated in appropriate extensions of temporal logic, which can express a real-time deadline, the probability that some event occurs, expected reward (for example, expected time or number of failures), or limit-average properties. Examples include probabilistic branching-time logics such as PCTL [56] and linear-time quantitative specifications which can express properties such as mean payoff [20]. Model checking for quantitative specifications draws on techniques for conventional model checking, such as graph-theoretic algorithms, symbolic approaches and automata-theoretic methods, but these must be combined with appropriate algebraic and numerical methods, e.g. the solution of linear equations to compute the probability. For example, for models with dense real-time a time-abstract model has to be constructed first (for a given property) and analysed using conventional probabilistic model checking techniques. As an alternative to the exact, exhaustive model checking algorithms, *statistical model checking* [103] is a way to establish if a property approximately holds by sampling and inspecting simulated executions of the system. For a chosen confidence level, one can deduce whether the property holds or not by estimating the proportion of executions that satisfy the property, or statistical inference.

Software model checking. Modelling typically involves a manual step of model derivation, in the native language of the appropriate model checker, either from the problem specification or from a protocol standard, for example Bluetooth [39]. Such effort is not sustainable in the long term, since it is time consuming, open to transcription errors, and assumes familiarity with the model checker used. In recent years, considerable effort has been put into the development of model checking directly from software, i.e. written in programming languages such as C or Java, including Java Pathfinder [99] and CProver [29]. This is based on the premise that state-transition system models can be extracted from software, which is possible with the help of compiler tools and abstract interpretation, and that – despite the state-space explosion problem – such extracted control flow graph models can be iteratively abstracted and refined, resulting in automatically establishing or refuting the property. The CEGAR (counter-example guided abstraction refinement) approach [31] is invoked, where counter-example traces are used to guide the refinement process, without user intervention. These techniques are becoming established in the software engineering community, for example the Astree project has successfully employed abstract interpretation to rule out floating point overflow errors in the Airbus controller software. However, although first steps have been made towards quantitative software verification (PASS and [65]), the technology is in its infancy, with limitations largely due to the difficulty of automated quantitative refinement and lack of a compositional approach to aid scalability.

Compositionality in verification. In the light of the state-space explosion problem, compositional frameworks that enable one to verify that the property holds for the composed system, based on an analysis of the components in isolation, are highly desirable as a means to achieve scalability. Compositional assume-guarantee frameworks have been extensively studied in the literature [91,30], and implemented and successfully applied in practice [83]. A variety of rule formats have been proposed, but their applicability is limited because such approaches rely on the user being able to provide the assumptions for each component of the system. Alternative approaches being pursued investigate methods for finding assumptions automatically [89]. More recent proposals focus on compositional verification for interface theories [38,80]. There is also a significant body of theoretical work which develops compositional reasoning of probabilistic systems, including for variants of probabilistic automata [95,36,27], but none of these were implemented in practice. The main difficulty is interference between probability and nondeterminism which may result in the failure of compositionality.

Runtime verification. Runtime verification is achieved through continually observing and intercepting the system parameters as it executes, performing fault detection, identification and reconfiguration [35], which makes this approach particularly appropriate for context-aware, adaptive systems. Various techniques have been developed that use model checking at runtime, see e.g. *models@runtime*, including steering to avoid failures in distributed systems [102]. Machine learning techniques have been applied to improve the quality of predictions based on the history of executions and for state estimation [97]. There is also increasing interest in incremental model checking techniques which have the potential to improve efficiency. However, these methods have not

until recently [44] been applied in the quantitative runtime verification setting.

Agent-based formalisms. Although traditionally process calculi formalisms have allowed a notion of a process or agent, these were rather weak, in that they did not have intentions, and instead they offered capabilities to the environment, and the environment (which could well be other agents) would attempt to synchronise and interact on these capabilities. In multi-agent systems [101], intentions are typically described by goals, which can be multi-objective and quantitative, for example achieving minimum start-up cost and maximum average price over the lifetime. Collaboration in agent communities is achieved through the sharing of common goals. Technically, negotiation and similar modes of agreement are a result of mechanism design that supports this kind of behaviour, though the situation complicates when issues such as privacy are considered. The potential of agent-based techniques to model ubiquitous computing systems has already been noted, for example for sensor fusion in ubiquitous computing. Some real-time goals have been considered, e.g. in the context of (controller synthesis for) timed automata. However, how best to incorporate a range of quantitative goals within the modelling formalisms, and how to design algorithms for the verification of collaborating communities of agents against quantitative multi-objective goals – particularly in the context of continuous and possibly stochastic dynamics – is a subject open to research.

Game-theoretic approaches. When considering communities of self-aware 'everyware', which aim to achieve certain objectives such as efficient management of their resources, one must also take into account their (possibly adversarial) interaction with environment. Game-theoretic approaches, which explicitly represent the moves and countermoves of players and adversaries, are a natural model to represent and analyse such behaviours through the study of winning strategies, for example minimisation of energy usage, and the respective trade-offs. Games have been extensively studied in computer science, with the work focusing mainly on the complexity of computing certain winning strategies, but we are also seeing the emergence of economic games in the context of coordination for sensor networks (for example in the analysis of the Aloha protocol [82]). The work extends to stochastic games [34] as well as real-time games, where the key difficulty is that the conventional region-based approaches to obtaining a finite-state quotient are no longer sufficient. An important use of real-time games is for controller synthesis [17], but models which feature the important combination of probability and real-time, necessary for Quality of Service, have not been considered to date. The majority of the cited work is theoretical, with few genuine ubiquitous computing scenarios, and henceforth considerable effort is need to put these techniques into practice.

3 Project Rationale

This VERIWARE project contributes to the science aspects of the the UK Ubiquitous Computing Grand Challenge [18], and focuses on one of its subgoals, that is, formulating automated verification techniques, methodologies and tools. The overriding desire is to prevent software faults in 'everyware' systems, thus ensuring their safety, dependability, performability and resource efficiency.

The summary of research to date presented above has identified key conceptual features of 'everyware' that any modelling effort must be able to address as a prelude to verification, such as quantitative goals, cooperation and competition mechanisms, discrete and continuous dynamics, and stochastic behaviour. The potential and promise of a range of automated verification techniques has also been amply illustrated, with some already applied in the context of ubiquitous computing. Yet, it is clear that the applicability of these techniques to ubiquitous computing is severely limited at present.

Encouraged by the success of model checking and its growing acceptance in industry, and, building upon quantitative model checking techniques, the central premise of this project is to enable 'everyware' verification through a shift towards employing the following:

- *Agent-based frameworks*. A faithful model of communities of 'everyware' must represent them as self-interested agents, guided by goals and incentives, and continuously interacting with the environment as well as other agents. The aim is to work with game-theoretic techniques and develop specification formalisms and automated techniques for quantitative analysis of cooperative behaviour.
- *Software-centric, component-based approach*. Manual model building effort is time-consuming and costly. The project seeks scalable automated frameworks for extraction and verification of models directly from real software, such as C, which will crucially depend on our ability to formulate efficient abstraction-refinement schemes. A secondary and more ambitious aim is to formulate effective synthesis (of models or software) from quantitative specifications. To achieve scalability, it will be essential to work with quantitative, compositional component-based frameworks.
- *Online methods*. The offline verification approach is unsuitable for highly dynamic, context-driven, scenarios typical for ubiquitous computing, where adaptation is dependent on a range of inputs and happens continuously, as the system executes. We will develop online techniques based on runtime verification, employing methodology based on statistical inference and/or machine learning. The intention is to combine model extraction with deployment and adaptation of models at runtime, based on system observations.

3.1 Key Research Questions

This project addresses a range of fundamental modelling and algorithmic questions related to 'everyware', with the main focus on the theory to practice transfer for 'everyware' verification. The research is guided by the following key scientific questions:

1. What are the appropriate models for 'everyware'?
2. What are the appropriate interaction, cooperation and competition mechanisms for 'everyware'?
3. What are the appropriate correctness criteria for 'everyware'?
4. What are the appropriate quantitative measures for 'everyware' adaptability, resource usage and dependability?
5. Is verified 'everyware' feasible in theory?
6. Is verified 'everyware' feasible in practice?

3.2 Project Objectives

The overall aim of this project is to achieve 'verified everyware' in the long term, to paraphrase the 'verified software' proposal [63] and the Verified Compiler challenge by Tony Hoare. Building on the successful theory to practice transfer for quantitative/probabilistic model checking, yet significantly advancing it in new and previously unexplored directions, the specific objectives of the project are to:

1. Investigate the fundamental principles, semantic models and frameworks necessary to support the effective future use of verification for 'everyware', with emphasis on the following aspects not covered by existing methods:
   - Autonomous adaptation
   - Resource constraints/guarantees
   - Cooperative/competitive behaviour
2. Formulate efficient algorithms and develop prototype implementations for:
   - Model extraction, adaptation and synthesis
   - Automated quantitative abstraction-refinement
   - Runtime quantitative verification
3. Perform a selection of case studies to test the effectiveness of the methods.

4 Preliminaries

We begin by giving a brief introduction to a selection of models and properties used in automated quantitative/probabilistic verification. The presentation is centred on probabilistic models and the probabilistic model checker PRISM [72] which forms the core of the project. All the models and specification formalisms discussed below are supported by PRISM, see [2].

4.1 Discrete Probabilistic Models

Probabilistic behaviour such as randomisation or occurrences of failure upon taking an action can be modelled using Markov chains. We use discrete discrete probability distributions to model transitions, quantifying the likelihood of moving to a given target state. Though simple, they are of fundamental importance because they often arise via discretisation from more complex models, such as those featuring continuous time or continuous probability distributions.

A *discrete-time Markov chain (DTMC)* D is a set of states S together with a transition probability matrix $\mathbf{P} : S \times S \to [0,1]$ such that $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ for all $s \in S$. We usually distinguish an initial state, $\bar{s} \in S$, and label states with atomic propositions $L(s) \subseteq AP$. The behaviour of a DTMC is represented by the set of its execution paths $s_0 s_1 s_2 \ldots$ such that $s_0 = \bar{s}$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A probability space can be defined over paths of the DTMC [11], where events correspond to measurable sets of paths, for example those reaching an error state. *Probabilistic reachability* then refers to the probability of reaching a given set of states and is a key concept for quantitative verification on which verification of more complex properties is based.

For systems that exhibit concurrency or underspecification in addition to probabilistic choice, we use *Markov decision process (MDP)* models M, which are given as a set of states S, a set Act of actions, and a partial transition probability function $\mathbf{P} : S \times Act \times S \to [0,1]$ such that $\sum_{s' \in S} \mathbf{P}(s,a,s') = 1$ for all $s \in S$, $a \in Act$. As for DTMCs, we distinguish the initial state $\bar{s} \in S$ and allow labelling with atomic propositions $L(s) \subseteq AP$. Execution paths of MDPs are sequences $s_0 a_0 s_1 a_1 s_2 a_2 \ldots$ such that $s_0 = \bar{s}$ and $\mathbf{P}(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 0$. Nondeterminism can be resolved by means of *adversaries*, which select one of the available actions upon reaching a state. Once an adversary is fixed, the behaviour of an MDP is fully probabilistic and the probability space formulated for DTMCs can be used to express the likelihood of events. In contrast to DTMCs, MDPs allows us to reason about the best- or worst-case system behaviour by quantifying over all possible adversaries. Probabilistic reachability for MDPs refers to computing the *minimum* or *maximum probability* of reaching a given set of states.

In order to reason about a broad range of *quantitative properties* we augment probabilistic models with *reward* (or *cost*) information. For the case of DTMCs (this is similar for other models) we endow a DTMC D with a *reward structure* $(\underline{\rho}, \iota)$, which consists of a vector of state rewards $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$, together with a matrix $\iota : S \times S \to \mathbb{R}_{\geq 0}$ of transition rewards, incurred each time a transition is taken. We allow both state and transition rewards for modelling convenience, as e.g. supported by PRISM, but note that transition rewards suffice. Given a reward structure, we can perform quanti-

tative analysis by computing *expectations* of (instantaneous or cumulative) rewards with respect to the probability space on paths, such as expected power consumption in the initial phase.

In quantitative verification, we use temporal logic PCTL [56, 12] to reason about path-based properties. We distinguish between state (Φ) and path (ϕ) formulas and allow path operators $\mathrm{X}\,\Phi$ (next state), $\Phi_1\,\mathrm{U}\,\Phi_2$ (until and its bounded variant $\mathrm{U}^{\leq k}$), as well as the usual derived operators $\mathrm{F}\,\Phi \equiv \mathit{true}\,\mathrm{U}\,\Phi$ (eventually) and $\mathrm{G}\,\Phi \equiv \neg\mathrm{F}\neg\Phi$ (always).

We are interested in computing the probability of set of paths from a given state satisfying a path formula, and likewise the expectation for some reward structure. Two operators are introduced for this purpose, the *probabilistic operator* $\mathrm{P}_{=?}\,[\,\cdot\,]$ and the *reward operator* $\mathrm{R}_{=?}\,[\,\cdot\,]$. $\mathrm{P}_{=?}\,[\,\phi\,]$ denotes the probability of a path formula ϕ satisfied in a given state of a DTMC. For reward structure $(\underline{\rho}, \iota)$, $\mathrm{R}_{=?}\,[\,\psi\,]$ denotes the expected reward for ψ, where ψ is a reward formula, for example $\mathrm{F}\,\Phi$, in which case the cumulated reward is calculated until state satisfying Φ is reached. For MDPs, we interpret these operators by universally quantifying over all adversaries, and hence distinguish between the minimum and maximum probability or expectation, e.g. $\mathrm{P}_{\max=?}\,[\,\cdot\,]$. We often use the bounded variants of the operators, e.g. $\mathrm{P}_{\bowtie p}\,[\,\cdot\,]$ where $p \in [0,1]$ is a probability bound and $\bowtie$ is a comparison operator, and similarly for the reward operator. The following are example specifications:

- $\mathrm{P}_{=?}\,[\,\mathrm{G}^{\leq 3600}\,\neg \mathit{fail}\,]$ - "the probability of no sensor failure occurring in the first hour" (DTMC);
- $\mathrm{P}_{\max\leq 0.01}\,[\,\mathrm{F}\,\mathit{lost}\,]$ - "the maximum probability, across all possible schedulers, of the sensor data being lost is no greater than 0.01" (MDP);
- $\mathrm{R}_{\min=?}\,[\,\mathrm{F}\,\mathit{reset}\,]$ - "the minimum expected number of data packets sent until system reset" (MDP), with reward 1 assigned to each state that sends a packet.

Model checking for PCTL proceeds in the way similar to CTL model checking, by induction on the syntax of the formula. The probabilistic operator involves a solution of a linear equation system for DTMCs and an iterative fixpoint computation (known as value iteration) for MDPs; the reward operator is similar, and is based on recursive equations or solving linear equations or linear programming problems. More expressive logics such as LTL can also be used, at a higher complexity cost. LTL model checking (for DTMCs/MDPs) is based on the automata-theoretic approach to non-probabilistic LTL and an exploration of the product construction in order to calculate the probability. For more details consult [45, 11].

4.2 Continuous-time Probabilistic Models

The models introduced in the previous section mark progress of time in discrete fashion, where each time step corresponds to taking a transition. In this section we briefly discuss models with continuous time and continuous probability distributions.

A widely used model that permits continuous time and transition delays modelled by (continuous) exponential distributions is *continuous-time Markov chains (CTMCs)*. It is given by a set of (discrete) states S and the transition rate matrix $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$. The rate $\mathbf{R}(s, s')$ determines the delay before the transition can occur, i.e. the probability of this transition being triggered within t time-units is $1 - e^{-\mathbf{R}(s,s')\cdot t}$, with the choice between more than one such transitions to be triggered in state s determined probabilistically. As for DTMCs, we distinguish an initial state $\bar{s} \in S$ and label states with atomic propositions $L(s) \subseteq AP$. Execution paths through a CTMC alternate states with real-valued time delays (residence time in a state). A probability space over paths can be defined [10] similarly to that for DTMCs, which allows us to reason about the probability of certain events occurring. CSL, the temporal logic for CTMCs, is similar to PCTL, except that it contains continuous versions of the time-bounded operators, such as $\mathrm{P}_{=?}\,[\,\mathrm{F}^{\leq t}\,\Phi\,]$, where t is real-valued, and a steady state operator $\mathrm{S}{=?}\,[\,\Phi\,]$ for computing long-run probability of residing in states satisfying Φ. Examples of CSL properties are:

- $\mathrm{P}_{=?}\,[\,\mathrm{F}^{\leq 2.5}\mathit{full}\,]$ - "the probability of sensor data buffer becoming full within 2.5ms";
- $\mathrm{S}_{=?}\,[\,\neg \mathit{fail}_A \wedge \neg \mathit{fail}_B\,]$ - "the long-run probability that wireless beacons A and B are operational".

CSL model checking for the probabilistic operator reduces to transient probability calculation, and typically proceeds through discretisation via uniformisation (a numerical transformation which optimises and numerically stabilises the computation of the transient probabilities). Model checking for the steady-state operator involves solving a linear equation system. Reward structures and operators can also be added; for more information, see [69].

CTMCs, like DTMCs, do not allow nondeterminism. Interactive Markov chains and continuous-time Markov decision processes feature exponentially distributed time delays in conjunction with nondeterminism, but are not supported by PRISM.

A model that is particularly appropriate in the context of distributed and service-based systems is based on *timed automata* [6], which allow real-time clocks and nondeterminism, and discrete probability distributions to represent randomisation and occurrence of failure. *Probabilistic timed automata* (PTAs) [76] can thus be viewed as MDPs extended with a set of real-valued clocks. A probabilistic extension of

the temporal logic TCTL can be used to express the properties of PTAs, which includes the probabilistic operator and certain forms of the reward operator dependent on the verification approach. The usual restriction of TCTL that clocks are compared to integer values also applies here.

Since PTAs allow dense real time, their semantics is an infinite-state MDP. There are two main approaches to model checking for PTAs. The first is a discretisation via *digital clocks* [74], which, for a (slightly) restricted class of PTAs, results in a finite-state MDP in which clocks can only take integer values. The other approach makes use of the symbolic representation of dense real-time in terms of *zones*, and proceeds through an adaptation of (forwards or backwards) algorithms for exploring the zone graph, while appropriately taking account of probabilities. The resulting abstraction is an MDP over zones, which can be verified using standard MDP techniques described above. An alternative approach is based on building stochastic game abstractions from MDPs over zones, and applying quantitative abstraction refinement [66].

Thus, model checking for PTAs essentially proceeds using techniques developed for MDPs and timed automata. More information can be found in [75].

We summarise by briefly mentioning types of models that naturally arise when modelling control of continuous processes. Such models contain continuous variables, generalising hybrid automata [58], as well as clocks, and include probabilistic hybrid automata [96] (which allow discrete probability distributions only) and stochastic hybrid automata [53] (which additionally contain continuous-time distributions and clocks). Model checking algorithms for these involve polyhedra methods and/or discretisation, and have high computational complexity.

4.3 Tools and Applications

Compared to conventional model checking, quantitative probabilistic model checking must judiciously combine graph-theoretic algorithms with numerical methods for calculating the probability or expectation. The main probabilistic model checkers are MRMC [64] (for DTMCs/CTMCs), LIQUOR [9] for MDPs and PRISM [72] (for DTMCs/CTMCs, MDPs and PTAs, together with the corresponding logics and reward extensions). Stochastic hybrid automata are supported by the software tool ProHVer [53]. PRISM is a symbolic model checker, and was implemented using a variant of Binary Decision Diagrams (BDDs), but it supports its own custom data structure and statistical model checking. The mentioned probabilistic model checkers have been applied to the analysis of a number of case studies from the ubiquitous computing domain, e.g. wireless communication protocols Zigbee and Bluetooth. However, all these concerned monolithic protocol descriptions and static configurations only.

5 Research highlights

In this section we present selected research highlights achieved in the initial phase of the VERIWARE project, followed by a brief summary of the remaining advances. They are all extensions to the formalisms and tools presented in Section 4.

5.1 Model Checking Cooperative Behaviour

The communities of 'everyware' – wirelessly connected and Internet-enabled – can be modelled using self-interested agents. It is well known that incentives such as positive or negative rewards have to be introduced to achieve cooperation between such agents, so as to increase their motivation and discourage selfishness. Traditionally, cooperative behaviour has been analysed using game theory; here, we are seeking a suitable framework in which we can represent cooperation schemes, state their common, quantitative goals, and automatically verify the cooperative behaviour. More concretely, the aim of the verification is to check whether the goal can be achieved, even in presence of selfish or cheating behaviour.

Probabilistic model checking with PRISM has already been used to study game-theoretic concepts, and more recently to analyse user-centric cooperative networks [5], but these studies employed DTMC/MDP models. We propose to use stochastic multi-player games as a model for cooperation, possibly in the presence of conflicting goals, which necessitates extending existing repertoire of models and quantitative verification techniques within PRISM. We also formulate a quantitative temporal logic which allows us to reason about the collective ability of a set of agents to achieve a goal expressed as a probability or expectation of some desirable event. The framework and implementation are reported in [26].

Stochastic games are a natural model for cooperative behaviour: similarly to MDPs, they admit discrete probabilistic distributions and nondeterministic choices, except that the latter are player-controlled. We work with turn-based games, rather than concurrent games [19]; we note that the latter are important to model incomplete information that is often encountered in distributed embedded settings. Formally, a (turn-based) *stochastic multi-player game (SMG)* G is given as the set of states S partitioned among a finite set Π of players, $(S_i)_{i\in\Pi}$, a finite nonempty set of actions Act (we assume $Act(s) \neq \emptyset$), and a partial transition probability function $\Delta : S \times Act \to Dist(S)$ where $Dist(S)$ denotes the set of probability distributions on S. As usual, we allow labelling of states with atomic propositions $L(s) \subseteq AP$.

We unfold the behaviour of a stochastic game G into *execution paths* $\lambda = s_0 a_0 s_1 a_1 \ldots$ that alternate states and actions. In each state $s \in S$, the choice of action from the available set $Act(s)$ is under the control of one player, namely, the player $i \in \Pi$ for which $s \in S_i$. Once action

$a \in Acts$ is selected, the successor state is chosen according to the probability distribution $\Delta(s, a)$. Similarly to adversaries for MDPs, we define a *strategy* for player $i \in \Pi$ as a function which selects one of the available actions in every state s. A *strategy profile* $\sigma = \sigma_1, \ldots, \sigma_{|\Pi|}$ comprises a strategy for all players in the game. Under a strategy profile σ, the behaviour of G is fully probabilistic and we can define a probability space over the set of all paths similarly to MDPs [21]. SMGs can also be endowed with (non-negative) state and transition-based rewards.

In order to express quantitative goals of SMGs, we formulate a probabilistic alternating time temporal logic with rewards (rPATL), which combines the alternating temporal logic ATL with the probabilistic logic PCTL. More concretely, we adopt the coalition operator $\langle\langle C \rangle\rangle$ of ATL [7], the probabilistic operator $\mathrm{P}_{=?}[\,\cdot\,]$ from PCTL [12], and a generalised variant of the reward operator $\mathrm{R}_{=?}[\,\cdot\,]$ from [45]. Players that cooperate form a coalition $\langle\langle C \rangle\rangle$, and we can analyse the coalition's behaviour under possibly adversarial behaviour of players not in the coalition ($\Pi \setminus C$). This allows us to reduce model checking of rPATL to an analysis of two-player stochastic games. Reward-based properties can be constructed using similar intuition; we omit the details of the different variants of reward operators from [26].

The following are typical quantitative goals that can be expressed in the logic rPATL:

- $\langle\langle\{1, 2\}\rangle\rangle \mathrm{P}_{\leq 0.01}[\mathrm{F}\, \mathit{fail}]$ - "players 1 and 2 have a strategy to ensure that the probability of failure occurring is at most 0.01, regardless of the strategies of other players";
- $\langle\langle C \rangle\rangle \mathrm{P}_{\max=?}[\mathrm{F}\, \neg \mathit{fail}]$ - "the maximum probability of coalition C having a strategy to ensure that failure is avoided, regardless of the strategies of other players";
- $\langle\langle C \rangle\rangle \mathrm{R}_{\min=?}[\mathrm{F}^{\infty}\, \mathit{stable}]$ - "the minimum expected energy that coalition C can conserve to ensure that a stable state is reached, regardless of the strategies of other players", for a reward structure that assigns energy to transitions taken.

Optimal probability of stochastic two-player games can be computed by algorithms similar to MDP value iteration [34]. Model checking for rPATL has high complexity, depending on the rewards operator used, but we nevertheless develop and implement efficient algorithms for computing probability and expectation to a desired precision based on fixpoint computation. We also demonstrate their usefulness in practice on a range of case studies, including collective decision making for sensor networks in presence of failure and team formation protocols, performing detailed quantitative analyses [26].

We briefly summarise the outcome of an analysis of a real-world demand-side management (MDSM) protocol. The original protocol is a microgrid that distributes energy to a number of households. The overall goal is to minimise energy usage. The distribution of tasks is probabilistic, and each household is rewarded for avoiding peak demand periods. The protocol has been validated through simulation but without considering selfish behaviour. Our model allows for selfish behaviour, and we have detected a flaw. This is because a household can selfishly deviate from the protocol without incurring any additional cost, which results in suboptimal performance. To correct the flaw, we introduce a punishment scheme. The results are shown in Fig. 1 (original, with flaw) and Fig. 2 (corrected). In the figures, the bold line shows all households following the original algorithm, while the dotted line shows the outcome without demand-side management. Horizontal dashes show deviations by collaborations of increasing size (shortest dash: individual deviation; longest dash: deviation of all households).

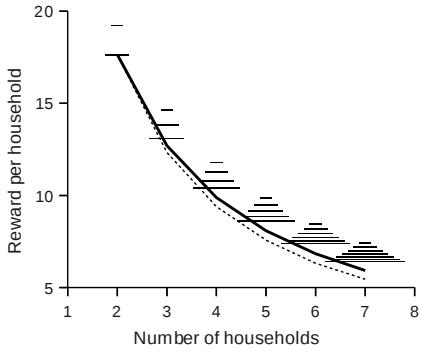

Fig. 1: Expected value per household for MDSM (original version).

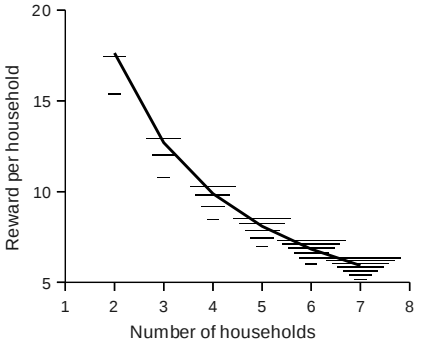

Fig. 2: Expected value per household for MDSM (version with punishment).

In future, we intend to develop further case studies, e.g. user-centric cooperation of [5], implement strategy generation, and formulate model checking algorithms for more complex, multi-objective goals.

5.2 Quantitative Verification for Implantable Devices

Embedded software is increasingly often used in medical devices, which monitor and control physical processes such as

electrical signal in the heart or dosage of insulin. Advances in sensor technology have resulted in millions of such devices in use today, working autonomously in closed loop or implanted in human body. Implantable medical devices must be designed to the highest levels of safety and reliability, yet errors in embedded software have led to a substantial increase in costly device recalls or even patient death. Automated verification is therefore called for to ensure dependability [51].

Probabilistic modelling and verification constitute a useful framework to enable quantitative analysis of risks and the impact of component failures and communication delays on implantable devices. A major challenge that must be addressed before this approach can be employed, however, is the need to incorporate *continuous dynamics* within quantitative, probabilistic models. The appropriate models in this context are variants of stochastic hybrid automata, e.g. [53], but they are highly complex and the existing analysis methods are limited.

Let us consider an implantable cardiac pacemaker device. A typical heart beats at 60-100 beats-per-minute (BPM). The sensors are placed on the heart muscle and are connected by wires to the processor that controls the heart. The sensors receive the electrical signal that passes through the heart, and the pacemaker is also able to generate such a signal, in case of missed heart beats. The activity of the heart can be monitored externally using an electrocardiogram (ECG) signal which can be described using non-linear ordinary differential equations. For example, the artificial ECG model introduced in [33] (see Fig. 3) generates an ECG signal (we only show the $\mathbf{x}$ coordinate which is sufficient for the work reported here):

$$\dot{\theta} = \omega, \qquad \dot{x} = -\sum_i \frac{\alpha_i^x \omega}{(b_i^x)^2} \Delta\theta_i^x exp\left[-\frac{(\Delta\theta_i^x)^2}{2(b_i^x)^2}\right] \qquad (1)$$

where we assume $\theta \in [-\pi, \pi]$ is the *cardiac phase*, $\Delta\theta_i^x = (\theta - \theta_i^x) \mathrm{mod}\, 2\pi$, $\omega = \frac{2\pi h}{60\sqrt{h_{av}}}$. In the above equation h is the instantaneous (beat-to-beat) heart rate in BPM and h_{av} is the mean of the last n heart rates (typically with $n = 6$) normalized by 60 BPM. The main advantages of this model is that the parameters can be adapted by learning from patient data to obtain a physiologically realistic model, and it also supports probabilistic modelling of switching between abnormal and normal heart behaviour, where the probabilities can also be adapted to individual patients. The drawback of the model is its non-linearity and complexity.

Jiang *et al* [62] developed a model-based framework for verifying real-time properties of cardiac pacemakers based on descriptions by Boston Scientific, a leading manufacturer. [62] develop a detailed model of a basic dual chamber pacemaker as a network of timed automata. They also model heart behaviour in terms of timed automata, and the composition of the pacemaker and the heart models can be analysed through simulation [61] and verification in UPPAAL [62]. They do not consider more general probabilistic, quantitative properties.

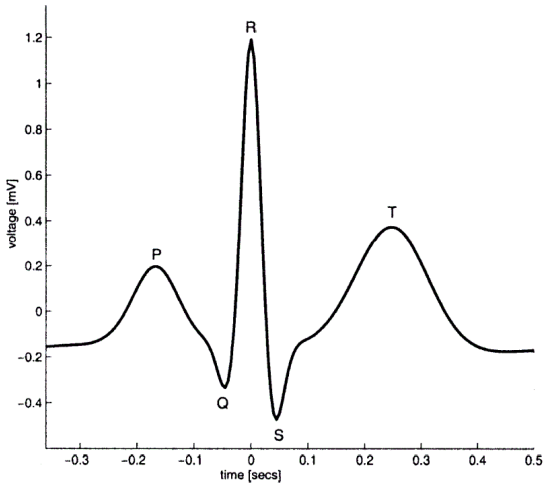

Fig. 3: Example electrocardiogram [84].

We enhance their effort by developing a methodology for quantitative, automated verification of pacemaker software models composed with a physiologically-relevant model of the heart based on the ECG equations shown above [24]. We convert the ECG signal to a sequence of action potentials, which correspond to the signal read on the sensors. We allow for both normal and abnormal heartbeats, as well as probabilistic switching between different heart conditions. We adopt, unchanged, the timed automata pacemaker model of [62]. To compose the heart and pacemaker models, we apply discretisation techniques: we use digital clocks [74] for timed automata. We formulate algorithms for quantitative, probabilistic analysis for properties such as whether the pacemaker corrects Bradycardia (slow heart beat) and does not induce Tachycardia (fast heart beat), and analyse the effect of undersensing (due to noise on sensor readings) and energy consumption based on a reward extension. We choose a discretisation step to ensure that the properties of interest are preserved. The framework is implemented using PRISM and MATLAB.

To illustrate the results, in Fig. 4 we show a graph obtained from composing the pacemaker with a heart that exhibits Bradycardia. The blue lines represent the behaviour of the faulty heart without the pacemaker. The green lines instead represent the behaviour of the faulty heart when equipped with the pacemaker. The x-axis shows the time at which events (beats) occur, whereas the y-axis shows the beat type. Note that the normal beats of the heart (blue lines, type 1 on y-axis) are slow, approximately one every two seconds. However, once the pacemaker is introduced, it induces an

atrium beat (type 3) which, after a slight delay, generates a normal ventricular beat (type 2), thus correcting the slow heart.

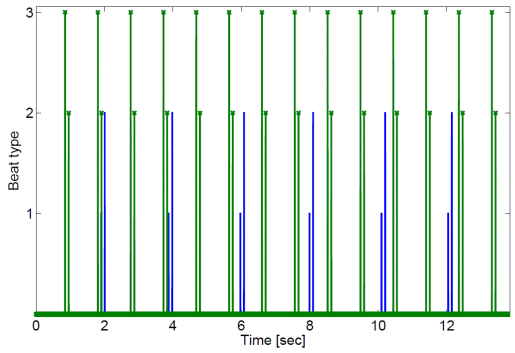

Fig. 4: Pacemaker correcting a faulty heart with Bradycardia.

More generally, the quantitative specifications that are applicable to implantable devices such as pacemakers and glucose monitors can be expressed in the logic MTL (Metric Temporal Logic) and include the following:

- $\Box_{[0,180]}(H(t) \geq 60) \wedge (H(t) \leq 100)$ - "in any interval of time up to time bound of 3 hours, there are between 60 and 100 heart beats per minute";
- $\Box_{[120,400]}(G(t) \geq 4) \wedge (G(t) \leq 10)$ - "the blood glucose level will settle to a normal range of 4 to 10 mmol/L 2 hours after a meal".

MTL has been applied to study risks of infusion pumps by modelling and analysing the insulin-glucose regulatory system via statistical model checking [93]. In future we intend to generalise the methods developed for model checking of MTL [23] and durational properties [25] over CTMCs to the case of cardiac pacemakers.

5.3 Quantitative Verification for Molecular Devices

Significant technological advances in programmable molecular assembly [92] have opened up the possibility to design and engineer sensor-enabled devices directly at the molecular level, to interface with biological systems. DNA molecules, in particular, are the subject of active research aimed at designing nanoscale logical circuits and robots, and numerous emerging applications are envisaged [67], several of which naturally extend the 'everyware' application domain. Unsurprisingly, given the safety-criticality of potential applications, there is already growing interest from within the software engineering community to establish a requirements engineering framework for molecular programming devices [81].

Programming of DNA circuits involves using short DNA strands as inputs to form information-processing circuits that proceed autonomously, binding and unbinding molecules, and producing other short strands as outputs. An established method to design such circuits is known as DNA strand displacement (DSD) [104]. DSD was given a process-algebraic semantics by Cardelli [16], which in turn formed the basis of a programming language and stochastic simulation environment also called DSD [90]. DSD designs are naturally modelled using biochemical reactions, and hence induce continuous time Markov chain models where stochastic rates are obtained from the kinetic rates. We can thus employ probabilistic verification against CSL properties. Molecular signalling pathways have been previously analysed using PRISM [57], see also [71]. Here we adapt the technology developed for this purpose to perform, for the first time, fully automatic quantitative verification of DSD designs [79], in the same way that verification tools are being applied to digital circuits.

The DSD programming language provides a textual notation for DNA circuit designs, which is mapped via its formal semantics to a system of chemical reactions. This is then exported (via SBML) to PRISM to perform verification. We analyse a cascade of transducer gates which had a known flaw. The flaw was discovered by manual inspection, but could not be found using stochastic simulation. It manifests itself as interference (or 'crosstalk') between two transducer designs, which means that the reactions proceed in an unintended manner: an input strand from one gate can react with the output strand of the downstream gate, producing reactive gates and causing unwanted deadlocks that cannot be reversed. We demonstrate that this flaw can be detected automatically, and a counterexample (trace to error) produced; we also show how to correct the design. In addition, we provide a range of *quantitative* property checks, including:

- $\mathrm{P}_{=?}[\,\mathrm{F}\, deadlock \wedge (reactiveGates = i)\,]$ - "the probability of deadlock occurring while there are i reactive gates in the cascade");
- $\mathrm{R}_{=?}[\,\mathrm{F}\, allDone\,]$ - "the expected time to completion" (with reward 1 assigned to each state in the model).

To illustrate the type of analysis performed, Fig. 5 shows the plot of probability over time T obtained for three different situations: (i) termination ($\mathrm{P}_{=?}[\,\mathrm{F}^{[T,T]}\, deadlock\,]$), (ii) termination in error ($\mathrm{P}_{=?}[\,\mathrm{F}^{[T,T]}\, deadlock \wedge (\neg allDone)\,]$), and (iii) termination with success ($\mathrm{P}_{=?}[\,\mathrm{F}^{[T,T]}\, deadlock \wedge allDone\,]$), where $allDone$ denotes the situation with the correct number of output strands and no reactive gates produced. Note that erroneous behaviour is more likely to arise initially, which can be explained by the fact that both conflicting reactions are irreversible. Thus, the occurrence of one determines the outcome, stabilising the probability plot.

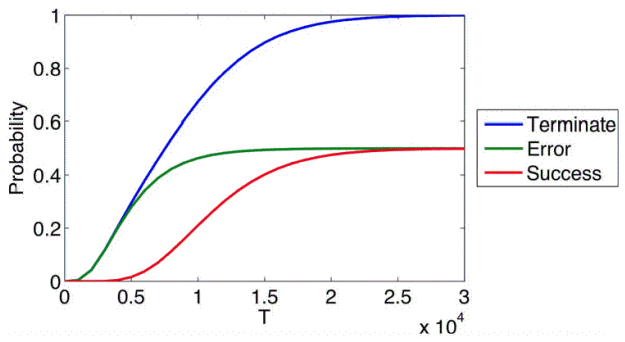

Fig. 5: Probabilistic analysis of a DNA transducer gate.

For more details about quantitative verification for molecular networks see [71]. Future work will include improving scalability (the system is limited to 6 molecules) and modelling more complex molecular self-assembly processes such as DNA origami and molecular walkers.

5.4 Compositional Quantitative Verification

The complexity of ubiquitous computing software – just consider the complexity of smartphone – means that achieving scalability of 'everyware' verification is a foremost goal. A promising approach to address the scalability of quantitative model checking is that of *compositional verification*, where the verification of a system composed from a number of components proceeds by analysing these in isolation, and inferring a global property without the need to construct the full state space. More specifically, we employ the *assume-guarantee* framework for verifying property G on a system $M_1 \| M_2$, which reduces to (i) checking that, under the *assumption* that some property A holds, M_2 is guaranteed to satisfy G, denoted $\langle A \rangle\, M_2\, \langle G \rangle$; and (ii) checking that M_1 always satisfies the assumption A under any context. Compositional assume-guarantee frameworks have been implemented for non-quantitative settings and successfully applied in practice [83].

A suggestion that compositional probabilistic verification can be achieved via *multi-objective* model checking was first presented in [41]. Given an MDP and a collection of, possibly conflicting, quantitative LTL formulas (that is, LTL formulas enclosed in a probabilistic or reward operator), multi-objective model checking aims to establish whether there exists an adversary satisfying all the properties. An example property is the existence of an adversary such that the probability of a sensor being operational is at least 0.9 and expected time to send data is at most 2ms. Formally, we work with quantitative multi-objective properties Ψ, which are conjunctions of predicates, denoted $[\phi]_{\geq p}$, each of which imposes a bound on probability or expectation of an ω-regular property ϕ being satisfied. Model checking for such properties can be reduced to solving an LP problem [73,46] or a variant of value iteration [47] which is significantly more efficient.

Based on the multi-objective probabilistic verification techniques developed in [41], we have formulated a compositional assume-guarantee framework for Markov decision processes [73] (more precisely, for a slight generalisation of MDPs known as probabilistic automata [95]). The approach is based on queries of the form $\langle \Psi_A \rangle\, M\, \langle \Psi_G \rangle$ interpreted as "under any adversary of M for which assumption Φ_A is satisfied, Ψ_G is guaranteed to hold". Both safety and liveness properties are supported, and several sound proof rules are developed, including asymmetric, circular and asynchronous. Below we give examples of rules for safety properties (omitting side conditions on synchronisation alphabets to simplify the presentation).

The first, simple rule is asymmetric:

$$\frac{\mathsf{M}_1 \models \Psi_A^{safe} \qquad \langle \Psi_A^{safe} \rangle\, \mathsf{M}_2\, \langle \Psi_G^{safe} \rangle}{\mathsf{M}_1 \parallel \mathsf{M}_2 \models \Psi_G^{safe}} \quad (\textsc{Asym-Safety})$$

The rule states that, given an appropriate assumption $\Psi_A^{safe} = [\phi_1]_{\geq p_1} \wedge \cdots \wedge [\phi_n]_{\geq p_n}$, we can check the correctness of a probabilistic safety property Ψ_G^{safe} on a two-component system, $\mathsf{M}_1 \| \mathsf{M}_2$, by performing one instance of (standard) model checking on M_1 (to check the first premise of rule (Asym-Safety)) and one instance of multi-objective model checking on M_2 (to check the assume-guarantee triple in the second premise). Since we avoid the construction of the full product state space, we can expect significant gains in terms of the overall performance.

The next, circular rule, where $\Psi_{A_1}^{safe}$, $\Psi_{A_2}^{safe}$, Ψ_G^{safe} are quantitative multi-objective properties comprising only safety properties:

$$\frac{\mathsf{M}_2 \models \Psi_{A_2}^{safe} \qquad \langle \Psi_{A_2}^{safe} \rangle\, \mathsf{M}_1\, \langle \Psi_{A_1}^{safe} \rangle \qquad \langle \Psi_{A_1}^{safe} \rangle\, \mathsf{M}_2\, \langle \Psi_G^{safe} \rangle}{\mathsf{M}_1 \parallel \mathsf{M}_2 \models \Psi_G^{safe}} \quad (\textsc{Circ-Safety})$$

reduces the verification to one instance of standard model checking on M_2 and two instances of multi-objective model checking on M_1 and M_2. More rules can be found in [26]. We have implemented the framework as an extension of PRISM [26] and demonstrated that compositional probabilistic verification can be superior to non-compositional verification on several benchmarks. Subsequently [42,43], we have also developed a method for automatically generating safety assumptions using learning, in conjunction with the asymmetric rule.

5.5 Quantitative Runtime Verification

The success of mobile devices such as smartphones and wearable 'everyware' heavily depends on services such as cloud computing and databases being autonomously managed by software. Since such software services must evolve and adapt, we argued in [14] that quantitative runtime verification is essential to achieve dependable software adaptation. When used at runtime, quantitative verification is able to predict and identify requirement violations, to plan the necessary adaptation steps, and to steer the system execution to comply with its (possibly changing) requirements.

The ability to provide continuous quantitative verification of software services poses great challenges. In [15], we proposed an extensive framework, called QoSMOS, for ensuring Quality of Service properties of service-based systems modelled as DTMCs. QoSMOS is based on autonomic computing and executes probabilistic verification tasks at appropriate points during system's execution, using the outcomes to plan the optimal selection of services according to the changing environment.

Recently [48], we tackled quantitative runtime verification for distributed systems which exhibit probabilistic behaviour, for example due to randomisation or failure, modelled as Markov decision processes. As an example consider a wireless network that supports mobile 'everyware' devices which can freely join and leave using the Zeroconf protocol. Upon joining, each device must determine a local IP address, which is dependent on the number of hosts in the network and the number of probes (query messages) that are broadcast before claiming a given IP address. A quantitative property of interest for this network is to minimise the probability of nodes choosing conflicting IP addresses.

Our framework [48] supports *incremental* quantitative runtime verification, which improves efficiency of verification tasks executed at runtime by reusing results from previous verification runs. We represent models *parametrically*, and allow for parameter changes during system execution. We formulate efficient techniques for both incremental model construction as well as incremental quantitative verification in presence of structural changes. This extends our earlier incremental probabilistic verification for MDPs which was limited to changes in probability values only [77].

The techniques have been implemented as an extension of PRISM [48] and evaluated on a range of examples, including JPL's Mars Exploration Rover modelled with probabilistic failure, on which we demonstrated a 10-fold improvement in efficiency. Currently, we work at the level of PRISM modelling language, but intend to extend to *real software* as, e.g., done in [65]. We will also employ techniques developed for parametric Markovian models [54, 44].

5.6 Summary of Further Advances

In addition to the results highlighted above, we have also progressed the research in the following directions:

- *quantitative abstraction-refinement*: we have formulated an abstraction-refinement framework for software with real-time, probabilities and data [70], such as SystemC programs, applied game-based abstraction to probabilistic hybrid systems [55], and in future intend to explore techniques developed for ProHVer [53];
- *quantitative synthesis*: we have solved the parameter synthesis problem for MDPs [54], devised a synthesis procedure for timed components from global specifications [28] based on the notion of quotient formulated for a compositional specification theory for components [22] inspired by interface automata [37], and in future intend to address controller synthesis from assume-guarantee specifications;
- *software tools*: we have developed and released various extensions of PRISM, including for probabilistic timed automata [72], approximate/statistical model checking [72], stochastic multi-player games for cooperative behaviour [26], multi-objective probabilistic verification using Pareto curves [47], and DSD to PRISM connection [79];
- *case studies*: we are developing a variety of case studies relevant for 'everyware', including energy-aware mobile protocols [50], smartgrid [26], collective decision making [26], implantable pacemakers [24], and molecular sensing devices at the nanoscale [79].

To complement these activities, the growing community of PRISM users has contributed a wide variety of case studies, many of which are relevant to 'everyware', for example analysing swarm robot behaviour, RFID protocols, cloud computing, assisted living systems, vehicle control, avionics, and many more; the interested reader is referred to the publications list at [2].

6 Challenges and Research Directions

We conclude with the remark that, despite substantial progress made in this project, due to the complexity of the ubiquitous computing scenarios and the need to interface to continuous processes, as well as to consider cooperation and mobility, the goals that we set ourselves – to achieve verified 'everyware' – are still highly challenging. We anticipate that following topics will be particularly difficult: effective synthesis from quantitative specifications; scalability and efficiency of abstraction-refinement schemes; efficiency and accuracy of approximate analysis; and software verification for wearable, implantable and mobile devices such as smartphones. We will measure success of the project by the

extent that the theoretical results obtained have influenced the practice.

References

1. http://newmediamonitor.wordpress.com/2011/06/08/a-fridge-that-tweets/.
2. PRISM website. www.prismmodelchecker.org.
3. U.S. Food and Drug Admin., List of Device Recalls. http://www.fda.gov/MedicalDevices/Safety/ListofRecalls/default.htm.
4. VERIWARE website. www.veriware.org.
5. A. Aldini and A. Bogliolo. Model checking of trust-based user-centric cooperative networks. In *AFIN 2012, The Fourth International Conference on Advances in Future Internet*, 2012. To appear.
6. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.
7. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
8. T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, M. O. Möller, P. Pettersson, C. Weise, and W. Yi. UPPAAL - Now, Next, and Future. In F. Cassez, C. Jard, B. Rozoy, and M. Ryan, editors, *Modelling and Verification of Parallel Processes*, number 2067 in Lecture Notes in Computer Science Tutorial, pages 100–125. Springer–Verlag, 2001.
9. C. Baier, F. Ciesinski, and M. Grosser. Probmela and verification of Markov decision processes. *SIGMETRICS Perform. Eval. Rev.*, 32(4):22–27, Mar. 2005.
10. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29:524–541, 2003.
11. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
12. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. FSTTCS'95*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.
13. D. Bucur and M. Kwiatkowska. On software verification for TinyOS. *Journal of Software and Systems*, 84(10):1693–1707, 2011.
14. R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, Sept. 2012.
15. R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimisation in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.
16. L. Cardelli. Strand algebras for DNA computing. In R. J. Deaton and A. Suyama, editors, *DNA*, volume 5877 of *Lecture Notes in Computer Science*, pages 12–24. Springer, 2009.
17. F. Cassez, J. Jessen, K. Larsen, J.-F. Raskin, and P.-A. Reynier. Automatic synthesis of robust and optimal controllers an industrial case study. In R. Majumdar and P. Tabuada, editors, *Hybrid Systems: Computation and Control*, volume 5469 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2009.
18. D. Chalmers, M. Chalmers, J. Crowcroft, M. Kwiatkowska, R. Milner, E. ONeill, T. Rodden, V. Sassone, and M. Sloman. Ubiquitous computing: experience, design and science: A grand challenge of the UKCRC. http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/, 2006.
19. K. Chatterjee, L. de Alfaro, and T. Henzinger. The complexity of quantitative concurrent parity games. In *Proc. SODA'06*, pages 678–687. ACM Press, 2006.
20. K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.
21. K. Chatterjee and T. A. Henzinger. A survey of stochastic omega-regular games. *Journal of Computer and System Sciences*, 2011.
22. T. Chen, C. Chilton, B. Jonsson, and M. Kwiatkowska. A Compositional Specification Theory for Component Behaviours. In H. Seidl, editor, *Proc. ESOP'12*, volume 7211 of *Lecture Notes in Computer Science*, pages 148–168. Springer-Verlag, 2012.
23. T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Time-bounded verification of CTMCs against real time specifications. In *Proc. 8th International Conference, Formal Modeling and Analysis of Timed Systems*, volume 6919 of *LNCS*, pages 26–42. Springer, 2011.
24. T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative verification of implantable cardiac pacemakers. In *Proc. 33rd IEEE Real-Time Systems Symposium (RTSS)*, 2012. To appear.
25. T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. Verification of linear duration properties over continuous-time Markov chains. In T. Dang and I. M. Mitchell, editors, *Proc. HSCC*, pages 265–274. ACM, 2012.
26. T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. In *Proc. 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *LNCS*, pages 315–330. Springer, 2012.
27. L. Cheung, N. Lynch, R. Segala, and F. Vaandrager. Switched probabilistic I/O automata. In *Proc. 1st International Colloquium on Theoretical Aspects of Computing (ICTAC'04)*, volume 3407 of *LNCS*, pages 494–510. Springer, 2004.
28. C. Chilton, M. Kwiatkowska, and X. Wang. Revisiting timed specification theories: A linear-time perspective. In M. Jurdzinski and D. Nickovic, editors, *Proc. FORMATS'12*, volume 7595 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2012.
29. E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In K. Jensen and A. Podelski, editors, *Proc. TACAS 2004*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.
30. E. Clarke, D. Long, and K. McMillan. Compositional model checking. In *Proc. 4th Annual Symposium on Logic in computer science*, pages 353–362. IEEE Press, 1989.
31. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *Proc. CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
32. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
33. G. D. Clifford, S. Nemati, and R. Sameni. An artificial vector model for generating abnormal electrocardiographic rhythms. *Physiological Measurement*, 31(5):595, 2010.
34. A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203 – 224, 1992.
35. J. Crow and J. Rushby. Model-based reconfiguration: Toward an integration with diagnosis. In *Proceedings, AAAI-91 (Volume 2)*, pages 836–841, jul 1991.
36. L. de Alfaro, T. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In K. Larsen and M. Nielsen, editors, *Proc. CONCUR'01*, volume 2154 of *LNCS*, pages 351–365. Springer, 2001.
37. L. de Alfaro and T. A. Henzinger. Interface automata.
38. L. de Alfaro and T. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, September 2001.
39. M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of Bluetooth device discovery. *Int. Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.

40. J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. In *Proceedings of the IEEE*, pages 127–144, 2003.
41. K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. In O. Grumberg and M. Huth, editors, *Proc. TACAS'07)*, volume 4424 of *LNCS*, pages 50–65. Springer, 2007.
42. L. Feng, M. Kwiatkowska, and D. Parker. Compositional verification of probabilistic systems using learning. In *Proc. 7th International Conference on Quantitative Evaluation of SysTems (QEST'10)*, pages 133–142. IEEE CS Press, 2010.
43. L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In D. Giannakopoulou and F. Orejas, editors, *Proc. FASE'11*, volume 6603 of *LNCS*, pages 2–17. Springer, 2011.
44. A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In R. N. Taylor, H. Gall, and N. Medvidovic, editors, *ICSE*, pages 341–350. ACM, 2011.
45. V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.
46. V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In P. Abdulla and K. Leino, editors, *Proc. TACAS'11*, volume 6605 of *LNCS*, pages 112–127. Springer, 2011.
47. V. Forejt, M. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In S. Chakraborty and M. Mukund, editors, *Proc. ATVA'12*, volume 7561 of *LNCS*. Springer, 2012. To appear.
48. V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. Incremental runtime verification of probabilistic systems. In *Proc. 3rd International Conference on Runtime Verification (RV'12)*, LNCS. Springer, 2012. To appear.
49. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. pages 258–273. Springer, 2005.
50. L. Gallina, T. Han, M. Kwiatkowska, A. Marin, S. Rossi, and A. Spano. Automatic energy-aware performance analysis of mobile ad-hoc networks. In *Proc. IFIP Wireless Days (WD'12)*, 2012. To appear.
51. A. O. Gomes and M. V. Oliveira. Formal specification of a cardiac pacing system. In *Proceedings of the 2nd World Congress on Formal Methods*, FM '09, pages 692–707, Berlin, Heidelberg, 2009. Springer-Verlag.
52. A. Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders, 2006.
53. E. Hahn, A. Hartmanns, H. Hermanns, and J.-P. Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, pages 1–42, 2012. To appear.
54. E. M. Hahn, T. Han, and L. Zhang. Synthesis for PCTL in parametric Markov decision processes. In *Proc. 3rd NASA Formal Methods Symposium (NFM'11)*, volume 6617 of *LNCS*. Springer, 2011.
55. E. M. Hahn, G. Norman, D. Parker, B. Wachter, and L. Zhang. Game-based abstraction and controller synthesis for probabilistic hybrid systems. In *Proc. 8th International Conference on Quantitative Evaluation of SysTems (QEST'11)*, pages 69–78. IEEE CS Press, September 2011.
56. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994. 10.1007/BF01211866.
57. J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 319(3):239–257, 2008.
58. T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–. IEEE Computer Society, 1996.
59. T. A. Henzinger and J. Sifakis. The discipline of embedded systems design. *Computer*, 40:32–40, 2007.
60. J. Hillston. *A Compositional Approach to Performance Modelling*. CUP, 1996.
61. Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam. Real-time heart model for implantable cardiac device validation and verification. In *ECRTS*, pages 239–248. IEEE Computer Society, 2010.
62. Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In C. Flanagan and B. König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 188–203. Springer, 2012.
63. C. Jones, P. O'Hearn, and J. Woodcock. Verified software: A grand challenge. *Computer*, 39:93–95, 2006.
64. J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Quantitative Evaluation of Systems (QEST)*, pages 243–244, Los Alamos, CA, USA, 2005. IEEE Computer Society.
65. M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In N. Jones and M. Muller-Olm, editors, *Proc. VMCAI'09*, volume 5403 of *LNCS*, pages 182–197. Springer, 2009.
66. M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.
67. K. L. Kroeker. The rise of molecular machines. *Commun. ACM*, 54(12):11–13, 2011.
68. M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. ESEC/FSE'07*, pages 449–458. ACM Press, September 2007.
69. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.
70. M. Kwiatkowska, G. Norman, and D. Parker. A framework for verification of software with time and probabilities. In K. Chatterjee and T. Henzinger, editors, *Proc. FORMATS'10*, volume 6246 of *LNCS*, pages 25–45. Springer, 2010.
71. M. Kwiatkowska, G. Norman, and D. Parker. *Symbolic Systems Biology*, chapter Probabilistic Model Checking for Systems Biology, pages 31–59. Jones and Bartlett, 2010.
72. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
73. M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In J. Esparza and R. Majumdar, editors, *Proc. TACAS'10*, volume 6105 of *LNCS*, pages 23–37. Springer, 2010.
74. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.
75. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, chapter Verification of Real-Time Probabilistic Systems, pages 249–288. John Wiley & Sons, 2008.
76. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

77. M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN-PDS'11)*, pages 359–370. IEEE CS Press, 2011.
78. M. Kwiatkowska, T. Rodden, and V. Sassone, editors. *From computers to ubiquitous computing, by 2020*, volume 366(1881), 2008.
79. M. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 9(72):1470–1485, 2012.
80. K. G. Larsen, U. Nyman, and A. Wasowski. Interface input/output automata. In *FM 2006*, volume 4085, pages 82–97. Springer, 2006.
81. R. R. Lutz, J. H. Lutz, J. I. Lathrop, T. Klinge, E. Henderson, D. Mathur, and D. A. Sheasha. Engineering and verifying requirements for programmable self-assembling nanomachines. In *ICSE*, pages 1361–1364. IEEE, 2012.
82. A. MacKenzie and S. B. Wicker. Stability of multipacket slotted aloha with selfish users and perfect information. In *INFOCOM*, 2003.
83. K. L. McMillan. A methodology for hardware verification using compositional model checking. *Sci. Comput. Program.*, 37(1-3):279–309, 2000.
84. P. McSharry, G. Clifford, L. Tarassenko, and L. Smith. A dynamical model for generating synthetic electrocardiogram signals. *Biomedical Engineering, IEEE Transactions on*, 50(3):289 –294, march 2003.
85. R. Milner. Ubiquitous computing: Shall we understand it? *The Computer Journal*, 49(4):383–389, 2006.
86. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.
87. G. Norman, C. Palamidessi, D. Parker, and P. Wu. Model checking probabilistic and stochastic extensions of the pi-calculus. *IEEE Trans. Software Eng.*, 35(2):209–223, 2009.
88. C. Palamidessi and O. M. Herescu. A randomized encoding of the pi-calculus with mixed choice. *Theor. Comput. Sci.*, 335(2-3):373–404, 2005.
89. C. S. Pasareanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008.
90. A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 2009.
91. A. Pnueli. Logics and models of concurrent systems. chapter In transition from global to modular temporal reasoning about programs, pages 123–144. 1985.
92. L. Qian and E. Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.
93. S. Sankaranarayanan and G. Fainekos. Simulating insulin infusion pump risks by in-silico modeling of the insulin-glucose regulatory system. In *Proc. CMSB'12*, LNCS. Springer, 2012. To appear.
94. S. Sarma, D. Brock, and K. Ashton. The networked physical world. Technical report, MIT-AUTOID-WH-0001, 1999.
95. R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.
96. J. Sproston. Decidable model checking of probabilistic hybrid automata. In M. Joseph, editor, *FTRTFT*, volume 1926 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2000.
97. S. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. Smolka, and E. Zadok. Runtime verification with state estimation. In S. Khurshid and K. Sen, editors, *Runtime Verification*, volume 7186 of *Lecture Notes in Computer Science*, pages 193–207. Springer Berlin / Heidelberg, 2012.
98. M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:327–338, 1985.
99. W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda. Model checking programs. *Autom. Softw. Eng.*, 10(2):203–232, 2003.
100. M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999.
101. M. Wooldridge. Intelligent agents. In G. Weiss, editor, *Multiagent systems*. MIT Press, 1999.
102. M. Yabandeh, N. Knezevic, D. Kostic, and V. Kuncak. Crystalball: predicting and preventing inconsistencies in deployed distributed systems. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 229–244. USENIX Association, 2009.
103. H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 23–39. Springer Berlin / Heidelberg, 2002.
104. B. Yurke, A. Turberfield, A. Mills, F. Simmel, and J. Neumann. A DNA-fuelled molecular machine made of dna. *Nature*, 406(6796):605–8, 2000.