Plasma Lab: A Modular Statistical Model Checking Platform

Axel Legay, Sean Sedwards, and Louis-Marie Traonouez(✉)

Inria Rennes – Bretagne Atlantique, Rennes, France
louis-marie.traonouez@inria.fr

Abstract. We present an overview of Plasma Lab, a modular statistical model checking (SMC) platform that facilitates multiple SMC algorithms, multiple modelling and query languages and has multiple modes of use. Plasma Lab may be used as a stand-alone tool with a graphical development environment or invoked from the command line for high performance scripting applications. Plasma Lab is written in Java for maximum cross-platform compatibility, but it may interface with tools and libraries written in arbitrary programming languages. Plasma Lab's API also allows it to be incorporated as a library within other tools.

We first describe the motivation and architecture of Plasma Lab, then proceed to describe some of its important algorithms, including those for rare events and nondeterminism. We conclude with a number of industrially-relevant case studies and applications.

1 Introduction

Statistical model checking (SMC) employs Monte Carlo methods to avoid the state explosion problem of probabilistic (numerical) model checking. To estimate probabilities or rewards, SMC typically uses a number of statistically independent stochastic simulation traces of a discrete event model. Being independent, the traces may be generated on different machines, so SMC can efficiently exploit parallel computation. Reachable states are generated on the fly and SMC tends to scale polynomially with respect to system description. Properties may be specified in bounded versions of the same temporal logics used in probabilistic model checking. Since SMC is applied to finite traces, it is also possible to use logics and functions that would be intractable or undecidable for numerical techniques. In recent times, dedicated SMC tools, such as YMER[1], VESPA, APMC[2] and COSMOS[3], have been joined by statistical extensions of established tools such as PRISM[4], UPPAAL[5] and MRMC[6]. In this work we describe

[1] www.tempastic.org/ymer/.
[2] http://archive.is/OKwMY.
[3] www.lsv.ens-cachan.fr/~barbot/cosmos/.
[4] www.prismmodelchecker.org.
[5] www.uppaal.org.
[6] www.mrmc-tool.org.

T. Margaria and B. Steffen (Eds.): ISoLA 2016, Part I, LNCS 9952, pp. 77–93, 2016.
DOI: 10.1007/978-3-319-47166-2_6

Plasma Lab[7], a modular Platform for Learning and Advanced Statistical Model checking Algorithms [5].

SMC approximates the probabilistic model checking problem by estimating the parameter of a Bernoulli random variable, for which there are well defined confidence bounds (e.g., [21]). The general principle is to simulate the model or system in order to generate execution traces. These traces are checked with respect to a logic such as Bounded Linear Temporal Logic (BLTL) [4] and the results are combined with statistical techniques.

BLTL restricts the classical Linear Temporal Logic by bounding the scope of the temporal operators. Syntactically, we have

$$\varphi, \varphi' := \ \mathsf{true} \mid P \mid \varphi \wedge \varphi' \mid \neg\varphi \mid X_{\leq t} \mid \varphi \, U_{\leq t} \, \varphi',$$

where φ, φ' are BLTL formulas, $t \in \mathbb{Q}_{\geq 0}$, and P is an atomic proposition that is valid in some state. As usual, we define $F_{\leq t}\varphi \equiv \mathsf{true} \ U_{\leq t}\varphi$ and $G_{\leq t}\varphi \equiv \neg F_{\leq t}\neg\varphi$. The semantics of BLTL, presented in Table 1, is defined with respect to an execution trace $\omega = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$ of the system, where each state (s_i, t_i) comprises a discrete state s_i and a time $t_i \in \mathbb{R}_{\geq 0}$. We denote by $\omega^i = (s_i, t_i), \ldots, (s_n, t_n)$ the suffix of ω starting at step i.

Table 1. Semantics of BLTL.

$\omega \models X_{\leq t} \, \varphi$ iff $\exists i, \ i = max\{j \mid t_0 \leq t_j \leq t_0 + t\}$ and $\omega^i \models \varphi$	
$\omega \models \varphi_1 \, U_{\leq t} \, \varphi_2$ iff $\exists i, \ t_0 \leq t_i \leq t_0 + t$ and $\omega^i \models \varphi_2$ and $\forall j, \ 0 \leq j < i, \ \omega^j \models \varphi_1$	
$\omega \models \varphi_1 \wedge \varphi_2$ iff $\omega \models \varphi_1$ and $\omega \models \varphi_2$	$\omega \models \neg\varphi$ iff $\omega \not\models \varphi$
$\omega \models P$ iff $s_i \models P$	$\omega \models \mathsf{true}$

Plasma Lab implements qualitative and quantitative SMC algorithms. Quantitative algorithms decide between two contrary hypotheses (e.g., is the probability to satisfy the requirement is above a given threshold), while quantitative techniques compute an estimation of a stochastic measure (e.g., the probability to satisfy a property).

The "crude" Monte Carlo algorithm is a quantitative technique that uses N simulation traces ω_i, $i \in \{1, \ldots, N\}$, to calculate $\tilde{\gamma} = \sum_{i=1}^{N} \mathbf{1}(\omega_i \models \varphi)/N$, an estimate of the probability γ that the system satisfies a logical formula φ, where $\mathbf{1}(\cdot)$ is an indicator function that returns 1 if its argument is true and 0 otherwise. Using the Chernoff-Hoeffding bound [21], setting $N = \lceil (\ln 2 - \ln \delta)/(2\varepsilon^2) \rceil$ guarantees the probability of error is $\Pr(\mid \tilde{\gamma} - \gamma \mid \geq \varepsilon) \leq \delta$, where ϵ and δ are the precision and the confidence, respectively.

The sequential probability ratio test (SPRT) of Wald [23] evaluates hypotheses of the form $\Pr(\omega \models \varphi) \bowtie p$, where $\bowtie \in \{\leq, \geq\}$. The SPRT distinguishes between two hypotheses, $H_0 : \Pr(\omega \models \varphi) \geq p^0$ and $H_1 : \Pr(\omega \models \varphi) \leq p^1$, where $p^0 > p^1$ and the test cannot work if $p^0 = p^1$. Hence, the SPRT requires

[7] https://project.inria.fr/plasma-lab/.

a region of indecision (an 'indifference region' [24]) which may be specified by parameter ϵ, such that $p^0 = p + \epsilon$ and $p^1 = p - \epsilon$. The SPRT also requires parameters α and β, which specify the maximum acceptable probabilities of errors of the first and second kind, respectively. An error of the first kind is incorrectly rejecting a true H_0 (a false positive); an error of the second kind is incorrectly accepting a false H_0 (a false negative). To choose between H_0 and H_1, the SPRT defines the probability ratio

$$ratio = \prod_{i=1}^{N} \frac{(p^1)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^1)^{\mathbf{1}(\omega_i \not\models \varphi)}}{(p^0)^{\mathbf{1}(\omega_i \models \varphi)}(1-p^0)^{\mathbf{1}(\omega_i \not\models \varphi)}},$$

where N is now the number of simulation traces generated *so far*. The test proceeds by performing a simulation and calculating *ratio* until one of two conditions is satisfied: H_1 is accepted if $ratio \geq (1-\beta)/\alpha$ and H_0 is accepted if $ratio \leq \beta/(1-\alpha)$. These thresholds are good approximations of the exact values that guarantee error probabilities α and β, improving as α and β approach zero [23].

2 Plasma Lab Architecture

One of the main differences between Plasma Lab and other SMC tools is that Plasma Lab proposes an API abstraction of the concepts of stochastic model simulator, property checker (monitoring) and SMC algorithm. In other words, the tool has been designed to be capable of using external simulators or input languages. This not only reduces the effort of integrating new algorithms, but also allows us to create direct plugin interfaces with standard modelling and simulation tools used by industry. The latter being done without using extra compilers.

The tool architecture is displayed in Fig. 1. The core of Plasma Lab is a lightweight controller that manages the experiments and the distribution mechanism. It implements an API that allows to control the experiments either through user interfaces or through external tools. It loads three types of plugins: 1. algorithms, 2. checkers, and 3. simulators. These plugins communicate with each other and with the controller through the API. Only a few classes must be implemented to extend the tool with custom plugins for adding new languages or checkers.

An SMC algorithm collects samples obtained from a checker component. The checker asks the simulator to initialize a new trace. Then, it controls the simulation by requesting new states, with a *state on demand* approach: new states are generated only when needed to decide the property. Depending on the property language, the checker either returns Boolean or numerical values. Finally, the algorithm notifies the progress and sends the results through the controller API.

Table 2 presents the list of simulator and checker plugins currently available with Plasma Lab. Plasma Lab has also been used to verify other types of models through a connection or an integration with other tools. Some of these case-studies are presented in Sect. 4.

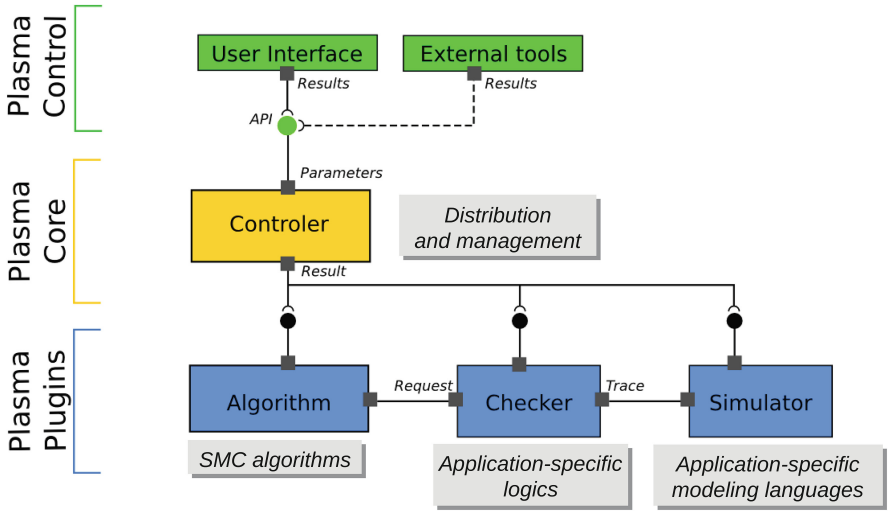

Fig. 1. Plasma Lab architecture.

Table 2. Plasma Lab plugins.

Simulators	
RML	Reactive Module Language: input language of the tool Prism for Markov chains models
RML adaptive	Extension of RML for adaptive systems
Bio	Biological language for writing chemical reactions
Matlab session	Allows to control the simulator of Matlab/Simulink
SystemC	Simulation of SystemC models. The plugin requires an external tool (MAG, https://project.inria.fr/pscv/) to instrument SystemC models and generate a C++ executable used by the plugin.
Checkers	
BLTL	Bounded Linear Temporal Logic
ALTL	Adaptive Linear Temporal Logic, and extension of BLTL with new operators for adaptive systems
GSCL	Goal and Contract Specification Language, a high level specification language for systems of systems
Nested	BLTL checker enhanced with nested probability operator
RML observer	A plugin that allows to write requirement as observers using a language similar to RML. It is used to write rare properties

Plasma Lab also includes several user interfaces capable of launching SMC experiments through the controller API, either as standalone applications or integrated with external tools:

- Plasma Lab Graphical User Interface (GUI). This is the main interface of Plasma Lab. It incorporates all the functionalities of Plasma Lab and allows to open and edit PLASMA project files.
- Plasma Lab Command Line. A terminal interface for Plasma Lab, with experiment and simulation functionalities, that allows to incorporate Plasma Lab algorithms into high performance scripting applications.
- Plasma Lab Service. A graphical or terminal interface for Plasma Lab distributed service. Its purpose is to be deployed on a remote computer to run distributed experiments, in connection with the Plasma Lab main interface.
- PLASMA2Simulink. This is a small "App" running from Matlab that allows to launch Plasma Lab SMC algorithms directly from Simulink.

2.1 Distributing SMC Experiments

Plasma Lab API provides generic methods to define distributed algorithms, which are a significant advantage of the SMC approach.

The distribution of the experiments is implemented with Restlet technology, using the architecture presented in Fig. 2. The main interface of Plasma Lab launches an SMC algorithm scheduler, while a series of services are launched on remote computers. Each service is loaded with a copy of the model simulator and a copy of the property checker. Then, the scheduler sends work orders to the services, via Restlet. These orders consist in performing a certain number of simulations and checking them with the checker. When a service has finished its work it sends the result back to the scheduler. According to the SMC algorithm, the scheduler either displays the results via the interface or decides that more work is needed.

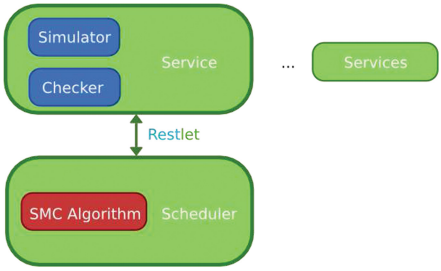

Fig. 2. Distributed architecture.

We have also implemented distributed SMC algorithms with Apache Spark. This alternative implementation allows to abstract even more the distribution mechanisms and facilitates the deployment of SMC experiments over large computing grids.

2.2 Tool Usage

We briefly present the usage of the tool. A more detailed description is provided on the website https://project.inria.fr/plasma-lab/documentation/. A generic usage of the tool GUI is presented in the flow diagram of Fig. 3. The GUI is

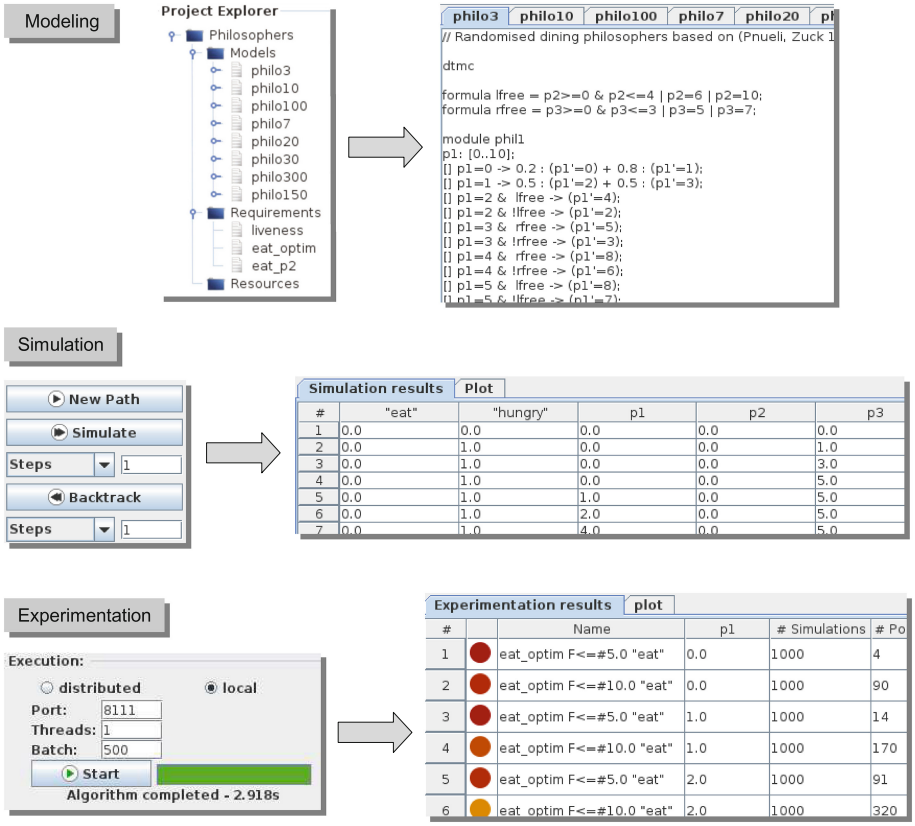

Fig. 3. Plasma Lab usage

composed of several panels that allow (*i*) to load, create and edit projects that comprise models and requirements, (*ii*) to perform simulations and debugging step-by-step, and (*iii*) to perform various forms of SMC experimentation and optimization, either locally or using distributed algorithms.

3 Plasma Lab SMC Algorithms

In addition to standard Monte Carlo and SPRT, Plasma Lab offers a number of advanced SMC algorithms for rare events simulation, nondeterminism optimisation and change detection.

3.1 SMC Algorithms for Nondeterminisitic Models

Markov decision processes (MDP) comprise probabilistic subsystems whose transitions depend on the states of the other subsystems. The order in which concurrently enabled transitions execute is nondeterministic and may radically affect

the probability to satisfy a given property or the expected reward. Since it is useful to evaluate the upper and lower bounds of these quantities, we are interested in finding the optimal *schedulers* that do this.

Memoryless schedulers have the complexity of the state space, while history-dependent schedulers have the complexity of the trace space of an MDP. Using hash functions and pseudo-random number generators, Plasma Lab encodes both memoryless and history-dependent schedulers as *seeds*, using $\mathcal{O}(1)$ memory. Each seed induces a Markov chain from an MDP, enabling Plasma Lab to find optimal schedulers using randomised algorithms with minimal memory.

The core of Plasma Lab's nondeterminism engine is its "simple sampling" algorithms [16]: a number of scheduler seeds are chosen at random and each induced Markov chain is verified using standard qualitative and quantitative SMC algorithms. Since the result of each sampling experiment has some probability of being incorrect, Plasma Lab implements confidence bounds modified for multiple schedulers [16].

Simple sampling has the disadvantage of allocating equal budget to all schedulers, regardless of their merit. To maximise the probability of finding an optimal scheduler with finite budget, Plasma Lab implements "smart sampling" algorithms [10,17], comprising three stages:

1. An initial undirected sampling experiment to approximate the distribution of schedulers and discover the nature of the problem.
2. A targeted sampling experiment to generate a candidate set of schedulers with high probability of containing an optimal scheduler.
3. Iterative refinement of the candidate set of schedulers, to identify the best scheduler with specified confidence.

Note that smart hypothesis testing may quit at any stage if an individual scheduler is found to satisfy the hypothesis with required confidence or if individual schedulers do not satisfy the hypothesis with required confidence but the average of all schedulers satisfies the hypothesis.

Stages 1 and 2 are based on the following formula for the probability of seeing a "near optimal" scheduler with a budget of $M \times N$ simulations:

$$(1 - (1 - p_g)^M)(1 - (1 - p_{\overline{g}})^N) \tag{1}$$

M is the number of schedulers and N is the number of simulations per scheduler. The values of p_g, the probability of seeing a near optimal scheduler, and $p_{\overline{g}}$, the average probability of the property using a near optimal scheduler, are unknown. Hence, since (1) is symmetrical, M and N are set equal in Stage 1. The results of Stage 1 provide approximations of p_g and $p_{\overline{g}}$, allowing the values of M and N to be chosen to approximately maximise (1) in Stage 2. Stage 3 applies simple sampling to the candidate set of schedulers produced by Stage 2. At each iterative step, the per-iteration budget is divided between the current candidate schedulers, SMC is applied and the least good half of the candidates are discarded. This refinement may continue until there remains only a single scheduler, so the initial value of N must be greater than or equal to the minimum number of simulations required to ensure the confidence of a single estimate.

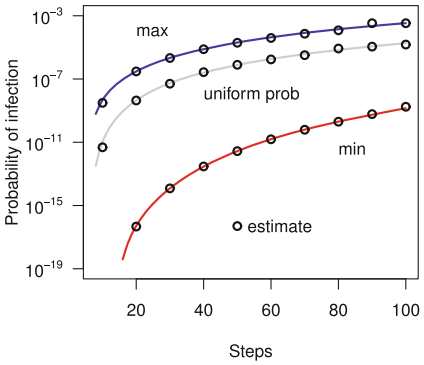

Fig. 4. Nondeterminism and rare events.

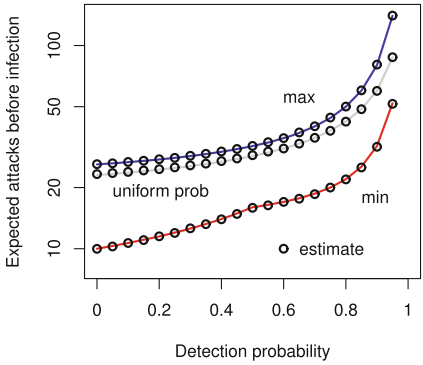

Fig. 5. Optimising rewards.

Figure 4 illustrates typical results for a virus infection model. The solid lines in Figs. 4 and 5 are the values calculated using numerical algorithms. The estimates of "uniform prob" confirm that our algorithms select uniformly from schedulers.

Costs or rewards may be assigned to the states and / or transitions of MDPs, so Plasma Lab also implements qualitative and quantitative smart reward estimation algorithms [17]. The algorithms consider *reachability rewards* (the expected cumulative reward over paths of a property with probability one), *cumulative rewards* (the expected cumulative reward of all path of a fixed length) and *instantaneous rewards* (the expected reward on a fixed step of all paths). Since in all cases no paths are rejected, $p_{\overline{g}}$ in (1) is effectively 1 and Stage 1 may be omitted, such that all the budget is directly assigned to M in Stage 2. Figure 5 illustrates typical results for a virus infection model.

3.2 Rare Event Simulation

Rare properties (i.e., with low probability) pose a problem for SMC because they are infrequently observes in simulations. Plasma Lab addresses this with the standard variance reduction techniques of importance sampling [11] and importance splitting [12–14].

Importance sampling works by weighting the probability distribution of the original system to favour the rare event. Since the weights are known, the correct result can be computed on the fly while simulating under the favourable importance sampling distribution. In addition to quantifying the probability of rare events in purely probabilistic systems, importance sampling can be useful when searching for optimal schedulers of nondeterministic systems whose optimal probability is low. E.g., importance sampling was used to generate the results shown in Fig. 4.

Importance splitting decomposes a property with low probability into a product of higher conditional probabilities that are easier to estimate. It proceeds

by estimating the probability of passing from one *level* to another, defined in Plasma Lab with respect to the range of a *score function* that maps states of the system × property product automaton to values. The lowest level is the initial state. The highest level satisfies the property. The initial states of intermediate simulations are the terminal states of simulations reaching the previous level.

The best performance is generally achieved with many levels of equal probability, requiring suitable score functions. Plasma Lab includes a "wizard" to construct *observers* in a reactive modules-like syntax from BLTL properties. The score function is defined within the observer and has access to all the variables of the system [14].

Table 3. Importance splitting results.

		Adaptive	Single	Parallel
Leader	Std. dev.	4.8×10^{-8}	1.3×10^{-7}	5.2×10^{-8}
	Levels	20	6	6
	Budget	1000	1000	5×1000
	Time (MC)	7.3s (30h)	2.5s (4.4h)	5.8s (5.0h)
Philosophers	Std. dev.	4.2×10^{-7}	7.7×10^{-7}	2.8×10^{-7}
	Levels	109	5	5
	Budget	1000	1000	5×1000
	Time (MC)	5.4s (2.3h)	1.7s (41m)	3.7s (1.4h)
Counters	Std. dev.	2.1×10^{-7}	5.0×10^{-7}	2.3×10^{-7}
	Levels	3942	4	4
	Budget	500	500	5×500
	Time (MC)	15s (7.5h)	2.8s (1.2h)	4.8s (1.9h)

Plasma Lab implements a fixed level algorithm and an adaptive level algorithm [13,14]. The fixed level algorithm requires the user to define a monotonically increasing sequence of score values whose last value corresponds to satisfying the property. The adaptive algorithm finds optimal levels automatically and requires only the maximum score to be specified. Both algorithms estimate the probability of passing from one level to the next by the proportion of a constant number of simulations that reach the upper level from the lower. New simulations to replace those that failed to reach the upper level are started from states chosen uniformly at random from the terminal states of successful simulations. The overall estimate is the product of the estimates of going from one level to the next.

The adaptive algorithm maximises variance reduction by minimising the number of simulations that fail at each level. This optimises performance on a single machine, but makes parallelisation inefficient. To take advantage of distributed computing, Plasma Lab therefore implements a parallel importance splitting algorithm based on the fixed level algorithm. We give performance results for various algorithms and models in Table 3. The adaptive algorithm outperforms the fixed level algorithm on a single machine, but the fixed level algorithm outperforms the adaptive algorithm in time and variance reduction when parallelised. All algorithms significantly outperform crude Monte Carlo (MC) [14].

3.3 Change Detection with CUSUM

Statistical techniques can also be used to perform runtime monitoring. The change detection problem consists in determining the occurrence of an event during the execution of the system, by looking at the variation of a probability measure along the execution. The CUSUM algorithm [3,22] has been used in signal theory to solve this problem. It computes a cumulative sum during the

execution that is compared to a sensitivity threshold. When this sum exceeds the stopping rule the algorithm determines that the expected event has occurred.

In Plasma Lab we have adapted this algorithm to SMC [19]. The idea is to observe the variation of the probability to satisfy a BLTL formula. In contrast to other SMC algorithms, the CUSUM algorithm only generates a single trace of the model. This trace is split in a set of samples taken at a regular time interval from the trace. Using this set of samples we can define the probability to satisfy a requirement at a certain time in the trace, by counting the number of samples that have satisfied the property. The CUSUM algorithm is then used to determine the time in the trace when this probability changes, which is the sign that an expected event has occurred.

Formally, we consider an execution $\omega = (s_0, t_0), (s_1, t_1), \ldots$ of the system and a BLTL property φ. We define a sequence of Bernoulli variables X_i such that X_i takes the value 1 iff $\omega^i \models \varphi$. We assume that we know the initial probability p_{init} of $\Pr(\omega \models \varphi)$. We want to observe a change of this probability such that $\Pr(\omega \models \varphi) > k$, with $k \in]0, 1[$. Like the SPRT, the CUSUM comparison is based on a likelihood-ratio test: it computes the cumulative sum S_n of the logarithm of the likelihood-ratios s_i over the sequence of samples $X_1, \ldots X_n$.

$$S_n = \sum_{i=1}^{n} s_i \qquad\qquad s_i = \begin{cases} \ln \frac{k}{p_{\mathsf{init}}}, & \text{if } X_i = 1 \\ \ln \frac{1-k}{1-p_{\mathsf{init}}}, & \text{otherwise} \end{cases}$$

The typical behaviour of the cumulative sum S_n is a global decreasing before the change, and a sharp increase after the change. Then the stopping rule's purpose is to detect when the positive drift is sufficiently relevant to detect the change. It consists in saving $m_n = \min_{1 \le i \le n} S_i$, the minimal value of CUSUM, and comparing it with the current value. If the distance is sufficiently great, the stopping decision is taken, *i.e.*, an alarm is raised at time $t_a = \min\{t_n : S_n - m_n \ge \lambda\}$, where λ is a sensitivity threshold.

4 Case Studies and Applications

In this section we present the different models and simulators that have been plugged with Plasma Lab, and used in case studies.

4.1 Systems of Systems: The DANSE Case Study

The DANSE[8] (Designing for Adaptability and evolutioN in System of systems Engineering) European project focuses on the development of a new methodology for System of Systems (SoS).

SMC techniques and Plasma Lab have been used within the project to analyse large heterogeneous systems like SoS. Plasma Lab has been integrated in the toolchain presented in Fig. 6. The SoS model is designed in UPDM (Unified Profile

[8] http://danse-ip.eu/.

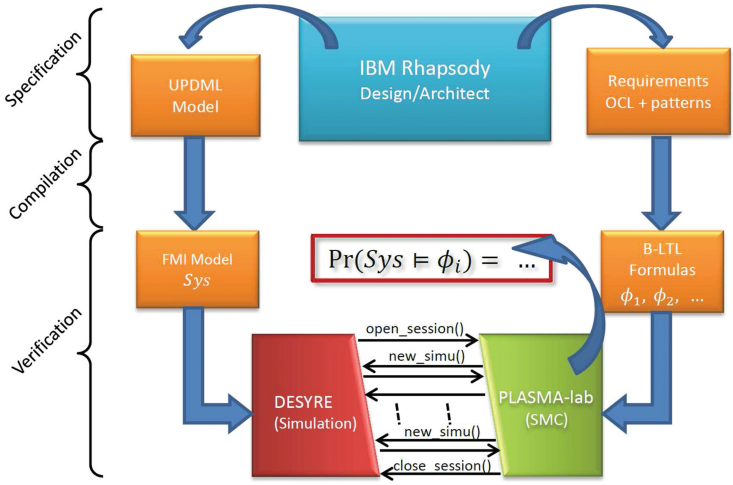

Fig. 6. DANSE methodology and toolchain.

for DoDAF/MODAF) with the tool IBM Rhapsody. Requirements are written with the Goal and Contract Specification Language (GCSL) and translated to BLTL. Plasma Lab SMC algorithms are used in combination with the tool DESYRE, developed by Ales, that simulates UPDM model using the FMI/FMU interface.

Goal and Contract Specification Language (GCSL). The DANSE project has introduced GCSL [2], a text-pattern based specification language with a formal semantics given by a temporal logic. This bridges the gap between natural language requirements and formal requirements. It is a combination of the Object Constraint Language (OCL) and the Contract Specification Language (CSL) developed in the SPEEDS project. CSL patterns are used to give a high-level specification of real-time components. They have been introduced to enable the user to reason about event triggering, that are equivalently replaced in DANSE by property satisfaction. The properties handled by these patterns are about the state of a SoS and we use OCL to specify these state properties. This language allows to build behavioural properties that express temporal relations about facts or events of the system. It is sufficiently powerful to describe precisely a state of a SoS. GCSL contracts can be translated to BLTL. Plasma Lab GCSL plugin allows to write requirement directly in GCSL.

Adaptive Reactive Module Language. Within the DANSE project we have also proposed [6] an extension of the RML language to describe stochastic adaptive systems (SAS). These systems consists in a set of components, organized with a certain topology, which we call a view. The composition of the system and its topology can evolve by changing its view. Views are represented by a combination of Markov chains modelled in the RML language. We introduce an extension

A-RML of the language to specify the sets of views of the system and to stochastic adaptive transitions between them (e.g. adding or removing components).

We also introduce the extensions A-BLTL and A-GCSL to reason about sequences of views in SAS. These new formalisms introduce new temporal operators to specify requirements about view change using assumptions and guarantees.

We have applied Plasma Lab to verify a case study of a SAS taken from the Concept Alignment Example (CAE) of the DANSE project. The CAE is a fictive adaptive system example inspired by real-world Emergency Response data to a city fire. It describes the organization of the firefighting forces in a city. We consider in our study that the city is initially divided into 4 districts, and that the population might increase by adding 2 more districts. Using SMC we verify that each view of the systems satisfy the requirements. Using A-GCSL contracts, translated to A-BLTL, we verify that the system is able to adapt its emergency answer in case the city expands.

4.2 Dynamic Motion Planning in DALi and ACANTO Projects

Plasma Lab has been integrated with robotic devices for the DALi[9] FP7 and ACANTO[10] H2020 projects, in the context of motion planning for assisted living [7,8]. Both projects rely on a novel online motion planning application of SMC to help those with impaired ability to negotiate complex crowded environments, such as museums and shopping malls. While DALi is focused on helping a single user reach a number of specific locations, ACANTO is concerned with therapeutic activities of groups of users, where group cohesion, social interaction and exercise are the metrics of interest.

The basic system architecture of our motion planner is illustrated in Fig. 7. Sensors, such as fixed cameras and cameras on robotic devices, locate fixed and moving objects in the environment. From this information a predictive stochastic model of human motion (the "social force model", SFM) is constructed, which is then used to generate plausible future trajectories of all the detected moving agents, given initial deviations from their current trajectories. Motion planning proceeds by hypothesizing different initial directions, then using Plasma Lab to estimate the probability that future trajectories will satisfy global constraints and objectives expressed in temporal logic. The best deviation is suggested to the user.

The operation of our motion planner for a single user (rectangular agent) is illustrated in Fig. 8. The solid red line denotes the direct path to the user's next local waypoint (green dot). The position and velocity of other pedestrians (circles) are indicated by vectors. With no modification, Plasma Lab estimates that the pedestrians will collide with high probability (not illustrated), but by making a deviation to the user's trajectory that diminishes over time (dashed

[9] www.ict-dali.eu.
[10] www.ict-acanto.eu.

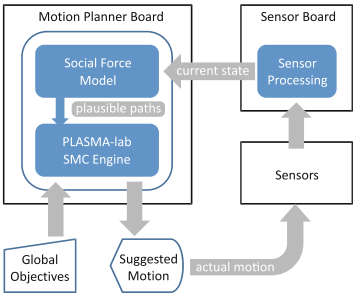

Fig. 7. Architecture of motion planner.

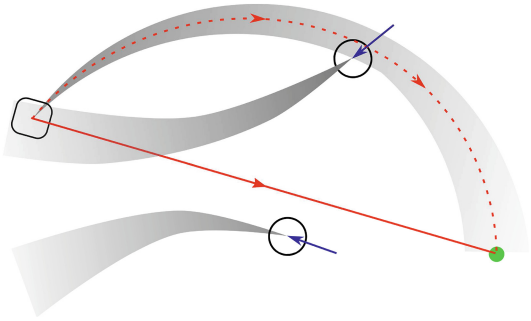

Fig. 8. Operation of motion planner.

red line), Plasma Lab predicts that the pedestrians will avoid each other with high probability (shaded areas).

To aid rapid development of a prototype algorithm, Plasma Lab was first integrated with MATLAB. The production algorithm was subsequently implemented on embedded hardware and finds the optimum trajectory in a fraction of a second.

4.3 Train Interlocking Systems

This case study has been analysed in collaboration with Université Catholique de Louvain and Alstom. We have analysed Braine l'Alleud station's interlocking system, a medium size railway station of the Belgian network. A representation of its track layout is shown on Fig. 9.

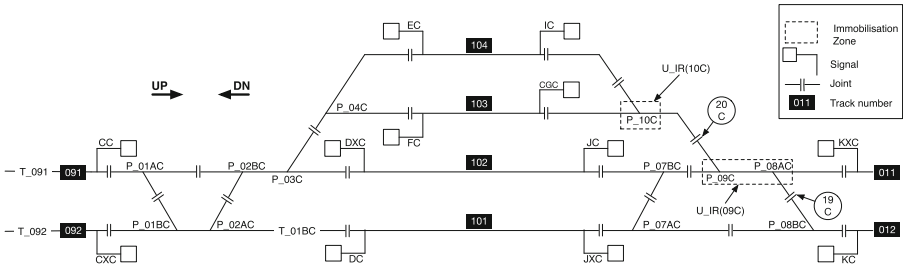

Fig. 9. Layout of Braine l'Alleud station.

Each station is composed of a set of physical components:

- The points (e.g. P_01BC) are the track components that guide the train from one track to another.
- The signals (e.g. CC) are the interface between the interlocking and the trains.

– The track segments (e.g. T_01BC) are the tracks where a train can be detected. They can be either occupied by a train or clear. On a station they are delimited each other by the joints.

A route is the path that a train must follow inside a station. A route is named according to its origin and destination point. For instance, Route R_CC_102 starts from Signal CC and ends on Track 102. A route can be set if it is reserved for a train or unset on the contrary. When a train is approaching to a station, a signalman will perform a route request to the interlocking in order to ask if the route can be set. The interlocking will handle this request and will accept or reject it according to the station state. To do so, an interlocking uses logical components like the subroutes or the immobilization zones, materializing the availability of some physical components. Such components are locked or released if they are not requested. Braine l'Alleud station is controlled by a unique interlocking composed of 32 routes, 12 signals, 13 track-circuits, and 12 points.

We verify two types of requirements: safety properties (e.g., avoid collisions of two trains on the same track), and availability properties (e.g., a route can always be eventually set). The verification process that we apply is described in Fig. 10. We use a simulator developed by Université Catholique de Louvain that is able to generate traces of the interlocking systems from a track layout and application data. This simulator is plug with Plasma Lab using a small interface developed with Plasma Lab's API. Then, the traces generated by the simulator are be used by Plasma Lab SMC algorithms to measure the correctness of the system. We have used Monte Carlo and importance splitting algorithms to verify this system.

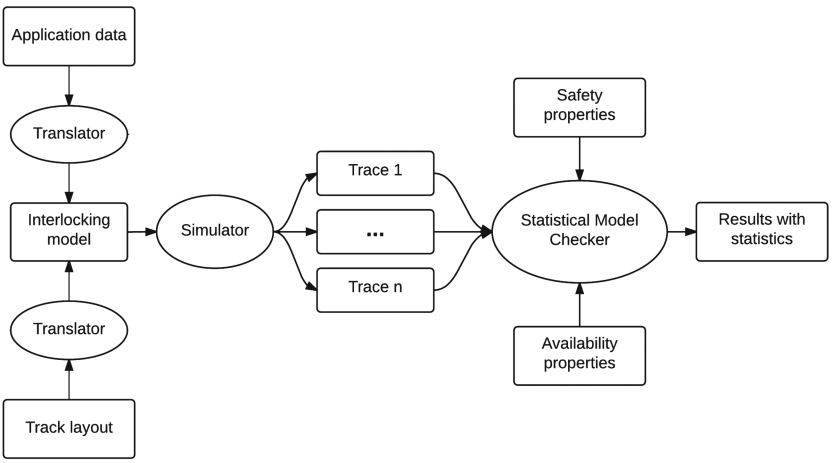

Fig. 10. Train interlocking verification steps.

4.4 Matlab/Simulink

Simulink models can be formally translated to hybrid automata [1] that interleave discrete state automata with complex dynamic behaviours described by differential equations. Model checking of these models is however undecidable. It is therefore interesting to use SMC to provide a formal analysis technique. Rather than translating Simulink models to a specific formal language, we have been able to directly interface Plasma Lab and Simulink [18]. We thus apply SMC algorithms by using the simulation engine provided by Simulink.

To achieve this we have developed a Matlab plugin for Plasma Lab, whose architecture is described in Fig. 11. It allows to control the Simulink simulator through the Matlab Control library[11]. It returns traces of observable variables to Plasma Lab SMC algorithms through the controller's API. Besides this plugin we have developed PLASMA2Simulink, a Matlab APP that provides a user interface to launch SMC experiments directly from Matlab.

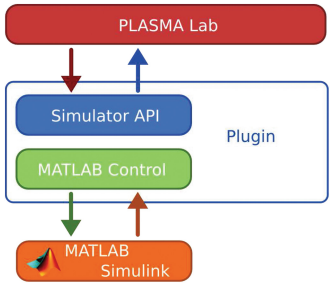

Fig. 11. Plasma Lab–Simulink interface.

We have used this plugin to analyse a Simulink model of a temperature controller of a pig shed. First, we use Monte Carlo techniques to verify that the controller maintains the temperature within comfortable limits, by activating fans and heaters. Second, by adding failures and wear to the system, we use the Plasma Lab CUSUM algorithm to determine the time when the controller becomes too inefficient – useful information that can be used to schedule maintenance of the system.

4.5 SystemC

SystemC is a high-level modelling language for specifying concurrent processes. It is implemented as a set of C++ classes that allow to perform event-driven simulation. Probabilistic behaviours can also be added.

We have implemented a SystemC plugin for Plasma Lab that is able to load a SystemC executable model and use it to generate simulations. Plasma Lab and the SystemC plugin is embedded in the toolchain of the Probabilistic SystemC Verifier (PSCV) tool [20]. The tool chain is presented in Fig. 12.

This toolchain has been used to analyse a SystemC model of an embedded control system, similar to [15], but with more components. The system consists of an input processor (I) connected to 50 groups of 3 sensors, an output processor (O), connected to 30 groups of 2 actuators, and a main processor (M), that communicates with I and O through a bus. At every cycle, 1 min, the main processor polls data from the input processor that reads and processes data from the sensor groups. Based on this data, the main processor constructs commands to be

[11] https://code.google.com/p/matlabcontrol/.

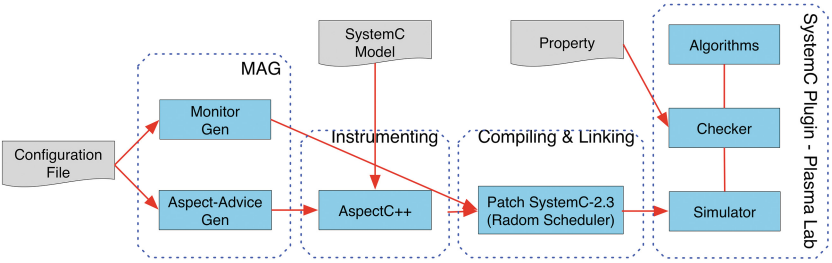

Fig. 12. Probabilistic SystemC Verifier toolchain.

passed to the output processor for controlling the actuator groups. The reliability of each component in the system is modeled as a Continuous-Time Markov Chain (CTMC) that is realized in SystemC. Using Plasma Lab we compute the probability of failure of each components.

5 Prospects

Our ongoing research is focused on the many interesting technical challenges arising from nondeterminism and continuous time [9]. In combination with our work on rare events, our longer term aim is to ensure Plasma Lab is able to address industrial scale verification problems in a way that is efficient and convenient for system engineers.

References

1. Agrawal, A., Simon, G., Karsai, G.: Semantic translation of Simulink/Stateflow models to hybrid automata using graph transformations. Electron. Notes Theor. Comput. Sci. **109**, 43–56 (2004)
2. Arnold, A., Boyer, B., Legay, A.: Contracts and behavioral patterns for sos: the EU IP DANSE approach. In: Proceedings of AiSoS. EPTCS, vol. 133, pp. 47–66 (2013)
3. Basseville, M., Nikiforov, I.V.: Detection of Abrupt Changes: Theory and Application. Prentice-Hall, Inc., Upper Saddle River (1993)
4. Biere, A., Heljanko, K., Junttila, T.A., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Logical Methods Comput. Sci. **2**(5) (2006). doi:10.2168/LMCS-2(5:5)2006
5. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: a flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40196-1_12
6. Boyer, B., Legay, A., Traonouez, L.-M.: A formalism for stochastic adaptive systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8803, pp. 160–176. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45231-8_12

7. Colombo, A., Fontanelli, D., Legay, A., Palopoli, L., Sedwards, S.: Motion planning in crowds using statistical model checking to enhance the social force model. In: IEEE Conference on Decision and Control (CDC), pp. 3602–3608 (2013)
8. Colombo, A., Fontanelli, D., Legay, A., Palopoli, L., Sedwards, S.: Efficient customisable dynamic motion planning for assistive robots in complex human environments. J. Ambient Intell. Smart Environ. **7**, 617–633 (2015)
9. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 99–114. Springer, Heidelberg (2016). doi:10.1007/978-3-319-33693-0_7
10. D'Argenio, P., Legay, A., Sedwards, S., Traonouez, L.: Smart sampling for lightweight verification of Markov decision processes. STTT **17**(4), 469–484 (2015)
11. Jegourel, C., Legay, A., Sedwards, S.: Cross-entropy optimisation of importance sampling parameters for statistical model checking. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 327–342. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31424-7_26
12. Jegourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 576–591. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_38
13. Jegourel, C., Legay, A., Sedwards, S.: An effective heuristic for adaptive importance splitting in statistical model checking. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8803, pp. 143–159. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45231-8_11
14. Jegourel, C., Legay, A., Sedwards, S., Traonouez, L.: Distributed verification of rare properties using importance splitting observers. In: ECEASST, vol. 72 (2015)
15. Kwiatkowska, M., Norman, G., Parker, D.: Controller dependability analysis by probabilistic model checking. Control Eng. Pract. **15**(11), 1427–1434 (2006)
16. Legay, A., Sedwards, S., Traonouez, L.-M.: Scalable verification of Markov decision processes. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 350–362. Springer, Heidelberg (2015). doi:10.1007/978-3-319-15201-1_23
17. Legay, A., Sedwards, S., Traonouez, L.: Estimating rewards & rare events in nondeterministic systems. In: ECEASST, vol. 72 (2015)
18. Legay, A., Traonouez, L.-M.: Statistical model checking of Simulink models with Plasma Lab. In: Artho, C., Ölveczky, P.C. (eds.) FTSCS 2015. CCIS, vol. 596, pp. 259–264. Springer, Heidelberg (2016). doi:10.1007/978-3-319-29510-7_15
19. Legay, A., Traonouez, L.: Statistical model checking with change detection. In: FOMACS (2016, to appear)
20. Ngo, V.C., Legay, A., Joloboff, V.: PSCV: a runtime verification tool for probabilistic SystemC models. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 84–91. Springer, Heidelberg (2016). doi:10.1007/978-3-319-41528-4_5
21. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Ann. Inst. Stat. Math. **10**, 29–35 (1959)
22. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1/2), 100–115 (1954)
23. Wald, A.: Sequential tests of statistical hypotheses. Ann. Math. Stat. **16**(2), 117–186 (1945)
24. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). doi:10.1007/3-540-45657-0_17