Recent Trends in Formal Validation and Verification of Autonomous Robots Software

Félix Ingrand

Recent Trends in Formal Validation and Verification of Autonomous Robots Software

Félix Ingrand
LAAS/CNRS
University of Toulouse
Toulouse, France
Email: felix@laas.fr

***Abstract*—The consequences of autonomous systems software failures can be potentially dramatic. There is no need to darken the picture, but still, it seems unlikely that people, insurance companies and certification agencies will let autonomous systems fly or drive around without requiring their makers and programmers to prove that the most critical parts of the software are robust and reliable. This is already the case for aeronautic, rail transportation, nuclear plants, medical devices, etc. were software must be certified, which possibly involve its formal validation and verification (V&V). Moreover, autonomous systems go further and embed onboard deliberation functions. This is what make them really autonomous, but open new challenges. We propose to consider the overall problem of V&V of autonomous systems software and examine the current situation with respect to the various type of software used. In particular, we point out that the availability of formal models is rather different depending on the type of component considered. We distinguish these different cases and stress the areas where we think we need to focus our efforts as to improve the overall robustness of autonomous systems.**

I. Introduction

Validation and Verification (V&V) of Autonomous Systems (AS)[1] software is not a "new problem". More than 20 years ago, some seminal work [Espiau et al., 1996] started to study how to guarantee robustness, safety and overall dependability of their software. Yet, for a number of reasons, the robotic and AI communities have mostly been focussed on other problems with respect to safe dependable autonomous robots. Meanwhile, there are other fields where bugs and errors can lead to catastrophic events (e.g. aeronautic, nuclear industry, rail transportation) where there is already a large corpus of research, but also successfully deployed tools and frameworks [Woodcock et al., 2009], whose goal is to improve the trust we can put in the software controlling these complex, although not autonomous, systems.

The fast and recent developments of autonomously driving cars have put the spotlight on the dramatic consequences of unverified software. Unfortunately, there is no doubt that autonomous vehicles will cause deadly accidents, but they will only become "acceptable" if the car makers have deployed all reasonably applicable and available techniques to ensure trust and robustness, and if as a result of these techniques, they outperform a regular human driver by one or two orders of magnitude. The decision to deploy these techniques may come from the car makers themselves, as a commercial argument to safety, or as an incentive from car insurances, or they may also comes from government certification agencies representing the general public concerns about safety (as this is already the case for aeronautic, railway, etc). In any case, we strongly believe that despite the somewhat human-biased argument that we are "all" good drivers, autonomous car will probably prevail if 5, 10 or 20 years from now, statistics show that they are indeed safer than regular human driven cars.

One original aspect to consider with respect to AS, is that unlike most critical systems in other domains, they exhibit and use deliberative functions (e.g. planning, acting, monitoring, etc) [Ingrand and Ghallab, 2017]. If one considers the models used by these deliberative functions, some are explicitly written by humans, while others are learned [Argall et al., 2009; Kober et al., 2013]. Similarly for functional components, somes also use learned models. We will see that if the explicit models are amenable to formal verification, the learned ones pose a new challenge to the V&V community.

For now, most of the trust we put in the AS Software (ASS) is acquired through test [Sotiropoulos et al., 2017; Koopman and Wagner, 2016]. Moreover, there are a number of "good" practices, architectures design, software development methodologies, model based techniques [Brugali, 2015; Mühlbacher et al., 2016], specification tools which all participate to establish this trust. Still, formal V&V, when applicable, has the potential to bring a level of confidence unreachable by other practices.

We also need to focus on approaches addressing realistic applications with real implementations and experiments. We already have reached the point where AS with tens of sensors and effectors, executing millions of line of code, running tens of programs on multiple CPUs, are being deployed. The time where one could illustrate an approach with an example, over simplifying reality, and limited added values to the field has passed. Proving that a lego mindstorm will not crash in the wall, thanks e.g. to its LTL generated controller, is not quite the same problem than showing that an autonomous car is not going to do the same. Moreover, we think the dependability of the software should be consider as a whole, looking at the complete system. When it comes to V&V, unless you take adapted protective/containment measures between

[1]We consider here autonomous systems at large, i.e. including autonomous robots, autonomous vehicles, drones, cyber physical systems, etc.

components, the overall system will be, at best, as strong as your weakest link/component, and the way they are being organized, communicate and share resources in the architecture is as critical as the components themselves.

There are some recent valuable surveys available, but the reading grid they propose is somewhat different from the one we develop here. Their coverage of the decisional components is limited [Luckcuck et al., 2018], or they are limited to the decisional components [Seshia et al., 2016], or they present a larger safety picture [Guiochet et al., 2017], while we want to focus on formal V&V. Still, they are a good source of information in this fast growing field.

Last, our perspective is definitely from a roboticist point of view. First, we want to rely as much as possible on automatic synthesis of models and code, and second we are aiming at proving properties which are "useful" for the ASS programmer. Can we guarantee that the plan produced by the planner is safe, that it will be properly executed by the acting system? Can the CPU resources available on the robots guarantee that all components will run at the specified rate? Can we guarantee that the robot will stop in time when an obstacle has been properly detected? that the initialization sequence will not deadlock, etc. Overall, starting from some real ASS implementations, what is the current status with respect to V&V and how can we improve it?

The paper is organized as follow. Section II presents the V&V models and techniques which we think are relevant to ASS, while section III reviews the various situation with respect to availability of formal models in an AS. We then present in section IV some of the robotic software specification frameworks which could be transform toward a formal model, and give an example of such an automatic transformation with GenoM and four formal frameworks as well as the V&V results we obtain. Section V presents our prospective on the subject and concludes the paper.

II. Formal Models and V&V

Our goal here is not to survey such a large field of research. We point the reader to [D'Silva et al., 2008; Woodcock et al., 2009; Bjørner and Havelund, 2014] for overviews of formal methods in software development. Formal methods use a mathematical model to analyse and verify part of a program. The key point here is that these mathematical models can then be used rigorously to prove properties of the modelled program. Of course, formal methods can cover various part of the program life cycle, from the specifications down to the code, here we mostly focus on approaches which are close to the deployed and running code.

One should note that none of these three recent surveys mention robots nor AS. Either they consider that they are not different from any other application domains (which is quite puzzling as AS exhibit deliberative functions not usually deployed in non AS), or that V&V of ASS poses some new challenges which have not yet been adressed.

A. Models and Methods

There are many formal models available, and none of them cover all the needs. Some models are grounded in simple yet powerfull primitives. automatas and state machine [Bohren and Cousins, 2010; Jónsson et al., 2006; Li et al., 2018] are often put forward as they easily capture the various states of the subsystems and their transitions. Petri net [Costelha and Lima, 2012; Lesire and Pommereau, 2018] are also often used, as they easily model coordination, together with their time extension, e.g. time Preti net [Berthomieu and Diaz, 1991]. Time is also at the core of Timed Automata widely used in UPPAAL, Kronos and more recently in the latest BIP version. Other models are provided as language defined at a higher level of abstraction, such as the synchronous system family, but can be translated to mathematical or logical representation. Such a category also includes temporal logic [Kress-Gazit et al., 2011], situation calculus [Levesque et al., 1997; Claßen et al., 2012] as well as interval temporal logic also deployed on robots in various components. There are also methods geared toward hybrid systems [Tomlin et al., 2003].

From a roboticist point of view, we should consider the ones which seems the most appropriate to represent the type of behavior we have to model and to prove the type of properties we want to check.

B. V&V Techniques

Among the various techniques available to the V&V community, we consider four different families.

State exploration and model checking, given a transition function, are working offline on the reachable state of the system from a given initial state. Sometimes, the space can be studied without being completely built, but overall these approaches tend to suffer from state explosion and often face scalability issues.

Logical inference on the other hand works offline by building an over approximation of the reachable states set as a logical statement (e.g. obtained by combining invariants of components). So the set is build by intention more than by extension. If these approach can potentially address the state explosion which breaks model checking, they face another problem as the logical invariants can be too general and too loosely fit the real reachable states set.

Statistical model checking (SMC) approaches, instead of exploring the complete state space, sample it using a probability model of the transition function, and thus evaluate properties to be verified with a resulting probability. Indeed, there are many properties which are desirable, but not required 100% of the time. For example, if your drone flight controller, running at 1 KHz, looses one cycle every one hundred cycles, the consequences are probably not as dramatic as if it looses one cycle every other cycle. SMC approaches allow one to "explore" states space whose size blows the regular state exploration techniques.

Last, *Runtime verification* is more an online approach where the state transition model is given to an engine which monitors and checks properties and consistency of the model on the fly.

This approach requires to specify what needs to be done when a property is violated, but allow verification of models which would not scale, nor fit with the two first approaches.

III. Autonomous System Software and Formal Models

With respect to the availability of formal models, we examine here the situation over the various software components of an AS.

A. Software Architecture

ASS needs to be organized along a particular chosen architecture framework. See [Kortenkamp and Simmons, 2008] for a survey of the different architectures available for AS. Yet, some architectures, by precisely defining how the layers are organized and how the components interacts, help/ease the V&V of the overall system. In any case, to keep things simple, we consider two layers: the Decisional Layer and the Functional Layer (see Fig. 1). In fact it boils down to two types of software components: i) those performing one of the deliberative functions, which often rely on models (e.g. planning actions, FDIR models, acting skills) which are then used to explore the problem space and find a solution (e.g. a planner combines actions models to find a plan, an FDIR component monitor the state of the system to find discrepancies with its model, etc), and ii) those performing some data and information processing to solve a problem or provide a service through regular algorithm implementing the function. The former usually involve heuristic searches in a space which cannot be reasonably explicitly computed, while the latter make the search for the solution explicit through the algorithms. The former may take an apriori unbounded amount of time, while the latter usually have a predictable global computation time. The former rely on high level abstraction models, while the latter are often programmed in classical programming language. Let us now examine the different situations with respect to the availability of models suitable for V&V.

B. Directly Programming with Formal Models

We consider here systems or software components which are programmed "directly" using some well established formal frameworks, which are presumably already used in other domains.

For example, the "synchronous approach" [Benveniste and Berry, 1991] and [Benveniste et al., 2003], its sister paper 12 year laters, has been instantiated in a number of languages (e.g. Lustre, Esterel, Signal, etc) and commercial frameworks. It has been deployed in critical domains such as, but not limited to, aeronautic (Scade, Lustre and Esterel), electronic (Signal). Esterel [Boussinot and de Simone, 1991] is used in [Simon et al., 2006] which presents a framework (Orccad) to deploy robotics systems programmed in the Maestro language

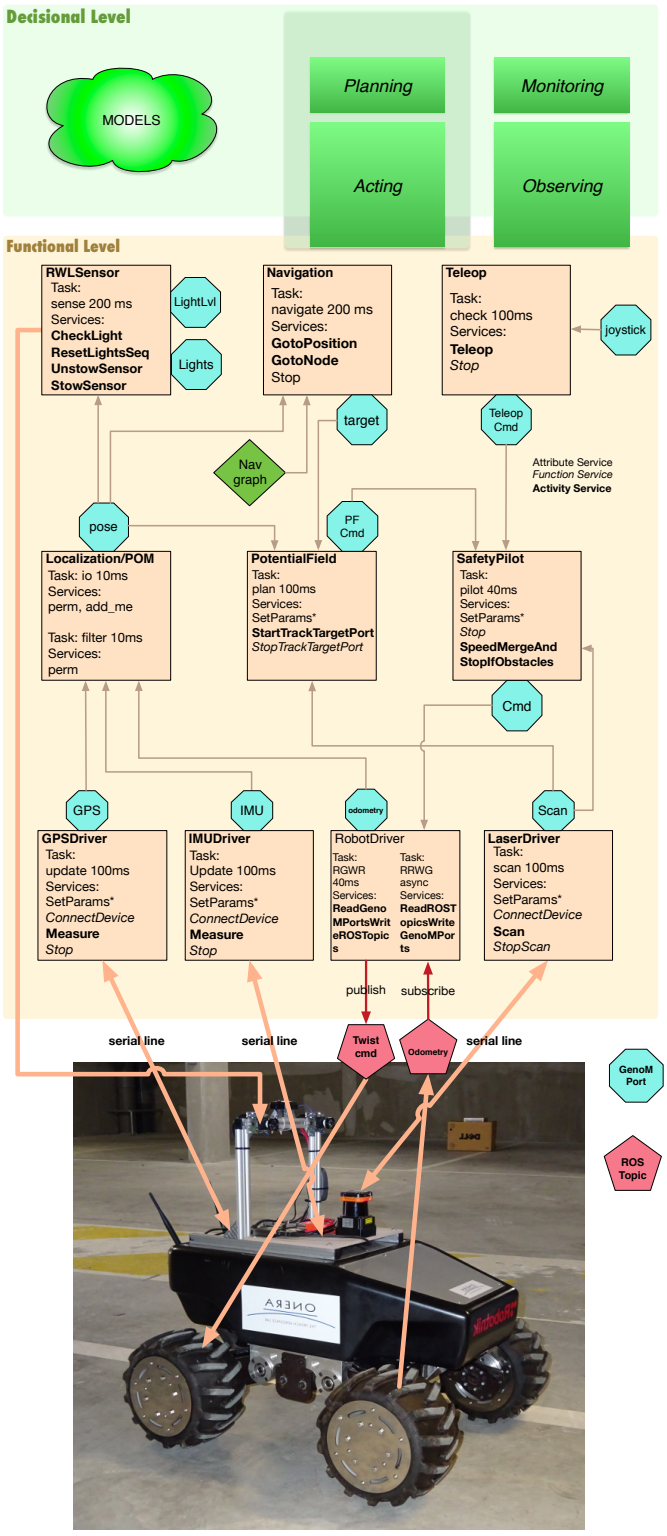

Fig. 1. Architecture of the OSMOSIS experiment[2].

(then translated in Esterel) to specify *Robot Tasks* and *Robot Procedures*. This was used to successfully program a complete AUV, and prove some properties on the system. Yet, the Maestro language was probably too abstract for roboticist, which did not adopt it.

[2]The OSMOSIS experiment is available at: https://osmosis.gitlab.io/. It provides 3 different implementations of the same setup using GenoM, MAUVE and pure ROS.

Similarly, there are decisional components which are programmed using situation calculus. Such a formalism has been used in Golog (for planning)[Levesque et al., 1997; Claßen et al., 2012] and Golex [Hähnel et al., 1998] (for acting) within a museum guide robot. However, the language is quite cumbersome, and was hardly been used for other deployments.

There are numerous other formalims available in the V&V community (e.g. BIP, LTL, UPPAAL, etc) which could have been used to program directly some robot components, but apart from toy examples, or localized and limited functionalities, this was not attempted. Similarly, there are formal tools to help the specification and the analysis of ASS (e.g. ALTARICA [Cassez et al., 2004]), but they are not easily linked to the real code running on the robot.

Overall, in this category we find some interesting attempt to bridge the gap between ASS and V&V, but clearly, the languages proposed and the required knowledge to make good use of them did not convinced the robotic programmers.

C. "Hidden" Formal Models

In this category we consider the software components which are programmed with models which have all the characteristics to be formal, but have not been presented as such and are not explicitly used with V&V methods.

For example, PDDL models widely used in automated *planning*, have all the right features to be formals. Even if its expressiveness can be somewhat limited when it comes to real robotics application, the semantic is clear and not equivoke, and most heuristic searches used to find plans, are correct (if not complete, nor optimal). Similarly, there are other planning formalisms (ANML, HTN, TLPlan, TalPLannner, temporal interval, etc) [Ingrand and Ghallab, 2017] which all fall in the same category. They where not designed for V&V, but they have all the right features to be used so. Some works, e.g. [Abdeddaïm et al., 2007; Bensalem et al., 2014], explicitly study the link between V&V and planning & scheduling.

If we consider the *acting* component, which is more interested in the operational model of how to execute an action (opposed to planning which is more on a model on how to use it) there are also many works. ASPiC [Lesire and Pommereau, 2018] is an acting system based on the composition of skill petri net, [Simmons and Pecheur, 2000] shows that TDL based acting components can be verified using NuSMV, RMPL [Williams and Ingham, 2003] (which rely on an Esterel like language) is also used for acting and monitoring, etc.

So there are many components (mostly deliberative within the decisional layer) which rely on some models which can be linked or transformed to some of the formal models used for V&V . Such transformation is seldom used, but is an option which could be easily activated if needed.

D. Learned Models

Learned models, by essence, are almost modelless. In an AS, one can learn a skill for *acting* (e.g. using reinforcement learning [Kober et al., 2013] or DBN [Infantes et al., 2010]), an action model for *planning*, (e.g. using MDP), or a perception classifier (e.g. using deep learning and convolutional neural network), but from a V&V point of view, these learned models are mostly black boxes.[3] Their popularity is due to the fact that they successfully tackle problems which resisted analytical modelling and solving.

Still there are some attempts to improve the dependability of learned models. In [Amodei et al., 2016], the authors identify five design pitfalls which can lead to "negative side effects" and they propose some guidelines as to prevent them. Unfortunately, none of them rely on formal methods, and if we can expect better models following them, there is no guarantee that false positive will not slip in. The authors of [Cicala et al., 2016] present three areas of ASS where they propose an automatic approach to do V&V. One of the area is Safe Reinforcement Learning for which they propose to deploy probabilistic model checking (see Section II-B) on discrete time model Markov chain. Similarly, the authors of [Seshia et al., 2016] identify five challenges, to formally verify system which use AI and Machine Learning. Probabilistic and randomized methods, as well as better environment modelling are among them. Overall, researchers are just starting to look at these issues. Yet, the presence, or not, of learned models in ASS will depend of their success. For now, we think that these models must be confined to non critical components, and if not, their results should be merged, combined and checked for inconstancy with others before participating to a decisional process.

An interesting work is proposed by [Feth et al., 2018] where they learn a model to help identify situations which require a higher level of awareness of the system.

E. No Model

In many situations, the code and the programs are written following some hopefully good programming practices, but overall, there exists no model at all of what it does. Still, there are a number of tools which make thorough checking of the code with static analysis and even some invariants extraction [D'Silva et al., 2008]. Even more, if formal V&V is really required, one can deploy an approach such as the one presented in [Täubig et al., 2011], which requires to annotate all the functions in the program with logical preconditions, assertions and effects which will then be checked and inferred by the formal tool (Isabelle). This is rather tedious, and can only be done by programmers which are familiar with the formal frameworks used *and* understand the algorithms being implemented. Nevertheless, the results are very encouraging.

F. Some Models

In this "vague" category, we consider all the components which are somehow specified with some languages, or model driven frameworks but which are not formal. For example, most robotic domain specific language DSL are seldom formal. One often use the term of "semi formal" in the sense that

[3]Note that if learning itself is considered a deliberative function, the learned models, can be used within different components, functional or decisional.

they have a clear syntax, but their semantics is not without any ambiguity, which prevent them to be directly fed to some V&V engine. There exist numerous framework to develop and deploy robotic softwares [Nordmann et al., 2016; Brugali, 2015]. Some offer specification languages, or rely on well known specification framework (e.g. UML, AADL, etc). Some just focus on providing tools and API libraries to ease integration of different components. Orocos [Bruyninckx, 2001] focuses on realtime control of robots. [Dhouib et al., 2012; Yakymets et al., 2013] presents RobotML a robotic domain specific language (Papyrus) and toolchain (based on Eclipse Modeling Project) which facilitates the development of robotics applications. Smartsoft [Schlegel et al., 2009] provides a framework to also specify the complete architecture of a robotic systems, while [Lotz et al., 2016] provide a meta model to separate user programmer concerns and system integrators issues . But if these tools greatly ease the overall architecture and analysis of the system, they remain short of connecting to a formal model.

G. Discussion

Overall, the situation is not completely hopeless, many components (in particular the deliberative ones) use formal models (hidden but present), which can be used for V&V. Providing the approach is sound, and that the models are valid w.r.t. the specifications, the results will be consistant w.r.t. the models. On the other end of the spectrum, code without any models, our main argument is that there should not be any. All the running code should be developed with some level of specification and structure to enable some verification. As for learned models, we need to consider a change of paradigm, and not so much prove that the models is validated and verified, but that it will be used and deployed in such a way that the trust we put in the system is not jeopardized.

Last, for components developed using a robotics framework for which a DSL can be derived toward a formal language, then one can also expect encouraging results. In the following section, we shall examine in detail this category.

IV. Leveraging Robotic Tools and Development Frameworks

Despite its success, ROS provides little support when it comes to ease V&V of the software with formal models. There are some efforts to model its communication layer [Halder et al., 2017], or to verify some simple properties [Come et al., 2018; Meng et al., 2015; Wong and Kress-Gazit, 2017], but overall, the lack of structure required to write ROS nodes makes it rather difficult to extract anything worth verifying. [Bardaro et al., 2018] proposes to model the robot software in AADL and then synthesize code in ROS. There are also systems which build some "run-time" verification of properties on the top of ROS [Huang et al., 2014; Sorin et al., 2016], but they hardly rely on ROS itself. [Kai et al., 2017] proposes model transformation using MontiArcAutomaton from a high level specification down to ROS code. Last we should point that the SMACH [Bohren and Cousins, 2010] component can be used to control ROS nodes with state machine models.

MAUVE [Gobillot et al., 2016; Doose et al., 2017] is an interesting framework which allows to extract temporal information from runs and verify then with LTL checker, but can also perform some runtime verification of temporal properties. Similarly, [Desai et al., 2017] proposes Drona, to perform model checking and runtime verification to check and enforce some other properties on a model using a Signal Temporal Logic. RoboChart [Cavalcanti, 2017; Miyazawa et al., 2017; Ribeiro et al., 2017] proposes an interesting approach where the programmer explicitly model the robotics application in a formal framework based on timed communicating sequential process. So the model is provided by the programmer not automatically extracted from the framework as we will see on an example in the next section.

We have seen that most robotic tools and frameworks do not provide any formal models per se, and if they do, it is usually up to the programmers to write both the program to run on the robot and the formal model.

In [Bjørner and Havelund, 2014], the authors write:

> *"We will argue that we are moving towards a point of singularity, where specification and programming will be done within the same language and verification tooling framework. This will help break down the barrier for programmers to write specifications."*

Following this advice, we advocate that the best way to introduce formal V&V in robotics component is to rely on existing robotic specification languages and frameworks and to offer some automatic translation to formal models. For this, we need to ensure that the semantics of the specification is correct, and is properly modelled in the targeted formalism.

A. The GenoM, RT BIP, Fiacre, UPPAAL example

GenoM [Mallet et al., 2010] is a tool to specify and implement robotic functional modules (see the Functional Layer on Fig. 1). These modules provide services in charge of a functionality which may range from simple low-level driver control (e.g. the LaserDriver or IMUDriver modules to control the Hokuyo LRF or the XSense IMU) to more integrated computations (e.g. Localizalion/POM for localization with an UKF, or PotentialField for navigation). GenoM proposes a language to completely specify the component down to (but not including) the C/C++ functions (codels) which implement the different stages/steps of the provided services [Foughali et al., 2018b]. This language fully specifies the shared ports (the green octagon on Fig. 1) between components (in and out), as well as the shared variables in a component, and the periodic tasks (i.e. threads) in which the services run. For each service, one defines the arguments (in and out), and the automata specifying the steps to follow to execute the codels, as well as their arguments. From a specification point of view, there is a clear semantics of what should be done and how it should be properly implemented.

A template mechanism allows to automatically synthesize the code for the component itself for various communication

Formal Frameworks	Offline	Online PocoLibs [Herrb, 1992]	Online ROS-Comm [Quigley et al., 2009]
RT BIP [Socci et al., 2013]	RT D-Finder [Ben Rayana et al., 2016]	RT BIP Engine [Abdellatif et al., 2010]	RT BIP Engine [Abdellatif et al., 2010]
FIACRE [Berthomieu et al., 2008]	Tina [Dal Zilio et al., 2015]	Hippo (under dev)	Hippo (under dev)
UPPAAL [Behrmann et al., 2006]	OK	NA	NA
UPPAAL-SMC [David et al., 2015]	OK	NA	NA

TABLE I
EXISTING FORMAL FRAMEWORK TEMPLATES FOR GenoM.

middleware (notably Pocolibs (shared memory) and RosComm (XML-RPC)). Either of this code is then linked to the codels library to provide the component itself.

Initially, GenoM was designed to help roboticists to develop functional components following a rigorous methodology but without having to worry about the OS and middleware specificities. But its rigorous specifications and the component template implementation allow us to develop templates to also automatically synthesize models for various formal languages [Foughali, 2018]:

a) RT BIP: is a framework[4] which allow modelling embedded real time systems, within components, including automata with guards (logical and temporal) and port for synchronization (rendez vous and broadcast) with other components. RT BIP is not to be confused with the original BIP template used in [Bensalem et al., 2011] which did not include time information. It can be used offline with RT D-Finder [Ben Rayana et al., 2016] to prove some properties. For this, it automatically extracts invariant, from the automatas, the interactions and the history clock and synthesizes an over approximation of the reachable states of the system. It then try to prove with a SAT solver that the desired property is satisfied in it. More interestingly, it can also be used online, and then use the RT BIP Engine [Socci et al., 2013] to runs the model itself, linked with the codels, and enforce the properties at run time. This runtime verification can also be augmented with more complex properties the roboticist may want to enforce.

b) UPPAAL/UPPAAL-SMC: is an integrated tool environment[5] for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types. Unlike BIP, it is using model checking to verify simplified TCTL (timed computation tree logic) properties in the modelled systems. The latest UPPAAL version (UPPAAL-SMC[6]) address the state explosion limit by offering a Statistical Model Checking extension. The timed automata can then be enriched with transition probabilities as to perform sampling of the reachable states consistant with the transition probabilities. Of course, the properties are then proven probabilistically true.

c) Fiacre: is a formal language[7] for specifying concurrent and real-time systems also based on automata (behavior), ports and transitions which can be guarded and sensitized over a time interval (similarly to time Petri nets). Note that the semantics is different from the timed automata used in BIP and UPPAAL.

So the same way GenoM synthesizes the components from specification and codels, it synthesizes formal models for these four frameworks. These templates includes temporal and statistical informations obtained by running the regular components with the proper probes. In particular, for all models, we include the extracted Worst Case Execution Time of the codels, as well as the number of state transition in the services automatas in the UPPAAL-SMC model.

The table I sums up the various formal framework for which GenoM templates are available, and the corresponding tools used. The resulting formal models are automatically synthesized for the experiment, such as the one presented Fig. 1. They are then enriched with a client model which specifies how the components would be used together (initialization sequence; perception, data fusion, motion planning and execution loop, etc). The complete model is then fed to the respective V&V tools which provide encouraging results [Foughali, 2018].

Using the FIACRE or UPPAAL models we can check offline than no **port** is ever read before it has been written at least once. In fact, this property was initially false because of a race condition in the initialization sequence of the robot where the NAVIGATION module may have tried to navigate before the first **pose** position of the robot had been produced by the LOCALIZATION/POM module. Similarly, we can compute the maximum time it takes between a *stop* request sent to NAVIGATION and the writing of the zero speed on the robot HW controller by ROBOTDRIVER from the **Cmd** port of SAFETYPILOT . Given the proper model of scheduler also written in FIACRE, we can check than the number of core on the CPU is sufficient or not [Foughali et al., 2018a]. Overall, even if model checking techniques suffer from state space explosion, the results obtained here on fairly complex robotic experiments are still encouraging. Online, using the BIP model with the BIP Engine, we capture temporal violation (e.g. task period or wcet values). We can also set monitors as to enforce new propreties. For example if the LASERDRIVER **scan** port has not been refreshed within 200ms, then a monitor stops the robot by explicitly calling the *Stop* service of the SAFETYPILOT module, which forces the *SpeedMergeAndStopIfObstacle* service to transit to its *stop* state and call the `StopSMAAO` codel which produces and writes a null speed in **Cmd**. Running the BIP Engine, in the OSMOSIS experiment, incurs a 15% increase in the CPU load, which remains acceptable considering the added safety.

[4]http://www-verimag.imag.fr/RSD-Tools.html
[5]http://www.uppaal.org/
[6]http://people.cs.aau.dk/~adavid/smc/
[7]http://projects.laas.fr/fiacre/

All these results are obtained with models automatically synthesized and automatic verification tools. One still needs to undestand how to express properties in the corresponding query language (e.g. LTL and patterns for Fiacre, TCTL for UPPAAL) and how to interpret the results, still, it is a big step forward in providing V&V tools to roboticists. To formally validate the obtained model, the semantics of GenoM has been first specified in Timed Transition Systems and then transformed in Timed Automata with Urgency and Data (See [Foughali et al., 2018a] for more details and proofs). The point being that whoever specify and implement components in GenoM, gets all these equivalent formal models for free, and can run the various V&V tools associated to them.

The GenoM formal frameworks templates should not be seen as the only possible path to infusing V&V in robotics. It is an example, of such a possible path, and there are numerous robotics frameworks which could do a similar automatic transformation of their specification toward formal models. We invite others to participate to such effort. With colleagues from ONERA and LAAS, we have setup an experiment, OSMOSYS[8], which proposes a robotic application (freely inspired from a CPSE Labs experiment) for a robot to inspect runway lights at night. The whole robotic experiment is currently available in pure ROS, MAUVE [Doose et al., 2017], and GenoM. A Gazebo simulator is also provided as to test the various implementations and analyze what can be extracted from the V&V tools being used. Other frameworks willing to contribute to this plateform are welcomed as to show with a fair presentation what can be done with each tool and at which cost (time, learning curves, etc). The only requirement is that each implementation must be effectively running on the simulation and the real robot.

V. Conclusion and Prospectives

In the proposed architecture, we distinguish between functional and decisional components. We have seen that some of these components already provides formal models (§III-C), so for these, the prospective seems to be more a problem of verifying the correctness of the search algorithms used. Some rely on DSL and specific frameworks (§III-F) which could be extended to automatically provides formal models on which one can perform V&V as we have shown in Section IV-A. We invite other robotic framework programmers to reach out and look at the possible formal frameworks they can connect to. Learned models (§III-D) have already been identified as outliers, so they probably need a "special" architectural setup for now and until we are satisfy with the confidence we can put in them. As for components with no model at all (§III-E), there are tedious solutions, if one cannot deploy them, one should at least consider reorganizing the code in such a way that it can rely on existing DSL or robotic frameworks.

On a different topic, the same way AI and robotics have to take into account human presence and interaction, V&V must also integrate it in the process. So we need to consider models of human behavior to introduce them in the V&V process. This can be models of the users of the AS itself (e.g. passenger of an autonomous car), but also models of people around the AS (e.g. pedestrian, or drivers of regular car). Of course this adds another layer of variability and expand again the size of the models to explore, but we should also consider this as an opportunity to "close" the model and keep the reachable states at a reasonable size.

Another topic which needs to be considered for is how these different models coexist and complete each other when it comes to proving a property over all the ASS. For example, the link between the different layer/component must also be verified. The communication and middleware should be properly modelled to be part of the V&V process. We also need to guarantee consistency over the various models deployed.

Last, we should keep in mind that specifications, even if they produce formally equivalent models which can be fed to V&V tools, needs to correctly capture the intents of the system designer. As pointed by [Rozier, 2016] "there is no escaping the 'garbage in, garbage out' reality". For this problem, we think that for now, we should rely on good old testing of the system as to check that specifications are correct and synthesize the proper formal model.

[8]https://osmosis.gitlab.io/

References

Abdeddaïm, Y., Asarin, E., Gallien, M., Ingrand, F., Lesire, C., and Sighireanu, M. Planning Robust Temporal Plans: A Comparison Between CBTP and TGA Approaches. In *ICAPS*, 2007.

Abdellatif, T., Combaz, J., and Sifakis, J. Model-Based Implementation of Real-Time Applications. In *EMSOFT*, 2010.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete Problems in AI Safety. *arXiv*, 2016.

Argall, B. D., Chernova, S., Veloso, M. M., and Browning, B. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.

Bardaro, G., Semprebon, A., and Matteucci, M. A use case in model-based robot development using AADL and ROS. In *ACM/IEEE Workshop on Robotics Software Engineering*, 2018.

Behrmann, G., David, A., and Larsen, K. G. A Tutorial on Uppaal 4.0. Technical report, Department of Computer Science, Aalborg University, Denmark, 2006.

Ben Rayana, S., Bozga, M., Bensalem, S., and Combaz, J. RTD-Finder - A Tool for Compositional Verification of Real-Time Component-Based Systems. In *TACAS*, 2016.

Bensalem, S., de Silva, L., Ingrand, F., and Yan, R. A Verifiable and Correct-by-Construction Controller for Robot Functional Levels. *JOSER*, 2011.

Bensalem, S., Havelund, K., and Orlandini, A. Verification and validation meet planning and scheduling. *STTT*, 2014.

Benveniste, A. and Berry, G. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 1991.

Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., and de Simone, R. The synchronous languages 12 years later. *Proceedings of the IEEE*, 2003.

Berthomieu, B. and Diaz, M. Modeling and Verification of Time-Dependent Systems Using Time Petri Nets. *IEEE Transactions on software Engineering*, 1991.

Berthomieu, B., Bodeveix, J.-P., Farail, P., Filali, M., Garavel, H., Gaufillet, P., Lang, F., and Vernadat, F. Fiacre: an Intermediate Language for Model Verification in the Topcased Environment. In *ERTSS*, 2008.

Bjørner, D. and Havelund, K. 40 Years of Formal Methods - Some Obstacles and Some Possibilities? *FM*, 2014.

Bohren, J. and Cousins, S. The SMACH High-Level Executive. *IEEE RAM*, 2010.

Boussinot, F. and de Simone, R. The ESTEREL Language. In *Proceeding of the IEEE*, 1991.

Brugali, D. Model-Driven Software Engineering in Robotics. *IEEE RAM*, 2015.

Bruyninckx, H. Open Robot Control Software: The OROCOS Project. In *ICRA*, 2001.

Cassez, F., Pagetti, C., and Roux, O. H. A Timed Extension for ALTARICA. *Fundam. Inform.*, 2004.

Cavalcanti, A. Formal Methods for Robotics: RoboChart, RoboSim, and More. In *Formal Methods: Foundations and Applications*. 2017.

Cicala, G., Khalili, A., Metta, G., Natale, L., Pathak, S., Pulina, L., and Tacchella, A. Engineering approaches and methods to verify software in autonomous systems. In *IAS*, 2016.

Claßen, J., Röger, G., Lakemeyer, G., and Nebel, B. Platas—Integrating Planning and the Action Language Golog. *KI-Künstliche Intelligenz*, 2012.

Come, D., Brunel, J., and Doose, D. Improving Code Quality in ROS Packages Using a Temporal Extension of First-Order Logic. In *ICRC*, 2018.

Costelha, H. and Lima, P. U. Robot task plan representation by Petri nets: modelling, identification, analysis and execution. *Autonomous Robots*, 2012.

Dal Zilio, S., Berthomieu, B., and Le Botlan, D. Latency Analysis of an Aerial Video Tracking System Using Fiacre and Tina. In *FMTV verification challenge of WATERS 2015*, 2015.

David, A., Larsen, K. G., Legay, A., Mikučionis, M., and Poulsen, D. B. Uppaal SMC tutorial. *STTT*, 2015.

Desai, A., Dreossi, T., and Seshia, S. A. Combining Model Checking and Runtime Verification for Safe Robotics. *RV*, 2017.

Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., and Ziane, M. RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications. In *SIMPAR*, 2012.

Doose, D., Grand, C., and Lesire, C. MAUVE Runtime: A Component-Based Middleware to Reconfigure Software Architectures in Real-Time. In *ICRC*, 2017.

D'Silva, V., Kroening, D., and Weissenbacher, G. A Survey of Automated Techniques for Formal Software Verification. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2008.

Espiau, B., Kapellos, K., and Jourdan, M. Formal verification in robotics: Why and how? In *ISRR*, 1996.

Feth, P., Akram, M. N., Schuster, R., and Wasenmüller, O. Dynamic Risk Assessment for Vehicles of Higher Automation Levels by Deep Learning. *arXiv*, 2018.

Foughali, M. *Formal Verification of the Functional Layer of Robotic and Autonomous Systems*. PhD thesis, LAAS/CNRS, 2018.

Foughali, M., Berthomieu, B., Dal Zilio, S., Hladik, P.-E., Ingrand, F., and Mallet, A. Formal verification of complex robotic systems on resource-constrained platforms. In *FormaliSE '18*, 2018a.

Foughali, M., Ingrand, F., and Mallet, A. GenoM3 Templates: from Middleware Independence to Formal Models Synthesis. *arXiv*, 2018b.

Gobillot, N., Guet, F., Doose, D., Grand, C., Lesire, C., and Santinelli, L. Measurement-based real-time analysis of robotic software architectures. In *IROS*, 2016.

Guiochet, J., Machin, M., and Waeselynck, H. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 2017.

Hähnel, D., Burgard, W., and Lakemeyer, G. GOLEX—bridging the gap between logic (GOLOG) and a real robot. In *KI Advances in Artificial Intelligence*, 1998.

Halder, R., Proença, J., Macedo, N., and Santos, A. Formal Verification of ROS-Based Robotic Applications Using Timed-Automata. In *IEEE/ACM International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, 2017.

Herrb, M. Pocolibs: POsix COmmunication LIbrary. Technical report, LAAS-CNRS, 1992.

Huang, J., Erdogan, C., Zhang, Y., Moore, B., and Luo, Q. ROSRV: Runtime verification for robots. In *Runtime Verification*, 2014.

Infantes, G., Ghallab, M., and Ingrand, F. Learning the behavior model of a robot. *Autonomous Robots*, 2010.

Ingrand, F. and Ghallab, M. Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 2017.

Jónsson, A. K., Verma, V., Pasareanu, C., and Iatauro, M. Universal executive and PLEXIL: engine and language for robust spacecraft control and operations. In *AIAA Space*, 2006.

Kai, A., Hölldobler, K., Rumpe, B., and Wortmann, A. Modeling Robotics Software Architectures with Modular Model Transformations. *JOSER*, 2017.

Kober, J., Bagnell, J. A., and Peters, J. Reinforcement Learning in Robotics: A Survey. *IJRR*, 2013.

Koopman, P. and Wagner, M. Challenges in Autonomous Vehicle Testing and Validation. *SAE Int. Jrnl. Transp. Safety*, 2016.

Kortenkamp, D. and Simmons, R. Robotic Systems Architectures and Programming. In *Handbook of Robotics*. 2008.

Kress-Gazit, H., Wongpiromsarn, T., and Topcu, U. Correct, Reactive, High-Level Robot Control. *IEEE RAM*, 2011.

Lesire, C. and Pommereau, F. ASPiC: an Acting system based on Skill Petri net Composition. In *IROS*, 2018.

Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 1997.

Li, W., Miyazawa, A., Ribeiro, P., Cavalcanti, A., Woodcock, J., and Timmis, J. From Formalised State Machines to Implementations of Robotic Controllers . In *Distributed Autonomous Robotic Systems*, 2018.

Lotz, A., Hamann, A., Lütkebohle, I., and Stampfer, D. Modeling Non-Functional Application Domain Constraints for Component-Based Robotics Software Systems. *arXivorg*, 2016.

Luckcuck, M., Farrell, M., Dennis, L., Dixon, C., and Fisher, M. Formal Specification and Verification of Autonomous Robotic Systems: A Survey. *arXiv*, 2018.

Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., and Ingrand, F. GenoM3: Building middleware-independent robotic components. In *ICRA*, 2010.

Meng, W., Park, J., Sokolsky, O., Weirich, S., and Lee, I. Verified ROS-Based Deployment of Platform-Independent Control Systems. In *NASA formal methods*. 2015.

Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., and Timmis, J. Automatic property checking of robotic applications. In *IROS*, 2017.

Mühlbacher, C., Gspandl, S., Reip, M., and Steinbauer, G. Improving Dependability of Industrial Transport Robots Using Model-Based Techniques. In *ICRA*, 2016.

Nordmann, A., Hochgeschwender, N., Wigand, D., and Wrede, S. A Survey on Domain-Specific Modeling and Languages in Robotics. *JOSER*, 2016.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. Y. ROS: an open-source Robot Operating System. In *ICRA*, 2009.

Ribeiro, P., Miyazawa, A., Li, W., Cavalcanti, A., and Timmis, J. Modelling and Verification of Timed Robotic Controllers. In *IFM*, 2017.

Rozier, K. Y. Specification - The Biggest Bottleneck in Formal Methods and Autonomy. In *VSTTE*, 2016.

Schlegel, C., Hassler, T., Lotz, A., and Steck, A. Robotic software systems: From code-driven to model-driven designs. In *ICAR*, 2009.

Seshia, S. A., Sadigh, D., and Sastry, S. S. Towards Verified Artificial Intelligence. *arXiv*, 2016.

Simmons, R. and Pecheur, C. Automating Model Checking for Autonomous Systems. In *AAAI Spring Symposium on Real-Time Autonomous Systems*, 2000.

Simon, D., Pissard-Gibollet, R., and Arias, S. Orccad, a framework for safe robot control design and implementation. In *CAR*, 2006.

Socci, D., Poplavko, P., Bensalem, S., and Bozga, M. Modeling Mixed-critical Systems in Real-time BIP. In *1st workshop on Real-Time Mixed Criticality Systems*, 2013.

Sorin, A., Morten, L., Kjeld, J., and Schultz, U. P. Rule-based Dynamic Safety Monitoring for Mobile Robots. *Journal Of Software Engineering In Robotics*, 2016.

Sotiropoulos, T., Waeselynck, H., Guiochet, J., and Ingrand, F. Can Robot Navigation Bugs Be Found in Simulation? An Exploratory Study. In *QRS*, 2017.

Täubig, H., Frese, U., Hertzberg, C., Lüth, C., Mohr, S., Vorobev, E., and Walter, D. Guaranteeing functional safety: design for provability and computer-aided verification. *Autonomous Robots*, 2011.

Tomlin, C. J., Mitchell, I., Bayen, A. M., and Oishi, M. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 2003.

Williams, B. C. and Ingham, M. D. Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers. *Proc. of the IEEE: Special Issue on Modeling and Design of Embedded Software*, 2003.

Wong, K. W. and Kress-Gazit, H. Robot Operating System (ROS) Introspective Implementation of High-Level Task Controllers. *JOSER*, 2017.

Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. S. Formal methods - Practice and experience. *ACM computing surveys*, 2009.

Yakymets, N., Dhouib, S., Jaber, H., and Lanusse, A. Model-driven safety assessment of robotic systems. In *IROS*, 2013.