Formal Specification and Verification of Autonomous Robotic Systems: A Survey

Matt Luckcuck, Marie Farrell, Louise Dennis,
Clare Dixon and Michael Fisher
Department of Computer Science, University of Liverpool, UK

29th June 2018

Abstract

Robotic systems are complex and critical: they are inherently hybrid, combining both hardware and software; they typically exhibit both cyber-physical attributes and autonomous capabilities; and are required to be at least safe and often ethical. While for many engineered systems testing, either through real deployment or via simulation, is deemed sufficient the uniquely challenging elements of robotic systems, together with the crucial dependence on sophisticated software control and decision-making, requires a stronger form of verification. The increasing deployment of robotic systems in safety-critical scenarios exacerbates this still further and leads us towards the use of formal methods to ensure the correctness of, and provide sufficient evidence for the certification of, robotic systems. There have been many approaches that have used some variety of formal specification or formal verification in autonomous robotics, but there is no resource that collates this activity in to one place. This paper systematically surveys the state-of-the art in specification formalisms and tools for verifying robotic systems. Specifically, it describes the challenges arising from autonomy and software architectures, avoiding low-level hardware control and is subsequently identifies approaches for the specification and verification of robotic systems, while avoiding more general approaches.

1 Introduction, Methodology and Related Work

Complex robotic systems are moving from an industrial setting into more commonplace scenarios, such as driverless cars [56], pilotless aircraft [155], and domestic assistants [153, 51]. With these new kinds of robotic systems comes the responsibility to ensure that they behave as intended, particularly when their failure can cause harm to humans. Various software engineering, testing, and simulation approaches have emerged for improving the safety of robotic systems; but formal methods provide much stronger assurance that these systems will behave as intended.

In this paper we contribute an overview and analysis of the current approaches to formally verifying the high-level decision making within robotic systems. Our decisions about the methodology of, and research questions answered by, our survey are outlined in § 1.1. § 2 describes some of the current non-formal approaches to engineering robotic systems, to provide the reader with the context of robotic systems development. The challenges that are faced when verifying a robotic system are discussed next: § 3 describes the challenges of the context of a robotic system (the *external* challenges) and § 4 describes the challenges of the organisation of the robotic system (the *internal* challenges). In § 5, we discuss the various formalisms used in the literature to overcome the identified challenges. In §6, we discuss the broad approaches used to verify robotic systems; covering techniques such as model checking, theorem proving, and runtime monitoring Finally, § 7 summarises our survey and presents our observations on the state-of-the-art in formal verification of robotic systems.

Ref	Formalisms/Tools Employed	Case Study or Example Used (where relevant)
[153]	SPIN model checker, Brahms modelling language (BrahmsToPromela translator), Multi-agent system, Linear Time Temporal Logic	Care-O-Bot robotic assistant in the Robot House Environment
[58]	Gwendolen for BDI agents, Agent JPF for checking agent properties, MCAPL, Linear Time Temporal Logic	Autonomous systems, RoboCup Rescue Scenario, Autonomous Satellite Scenario, autonomous pilotless aircraft scenario
[97]		Formally verified control algorithms for surgical robots
[11]	Reachability analysis	A wrong-way driver threatens two autonomously driving vehicles on a road with three lanes
[95]	PCTL models of Probabilistic State Machines, analysed in Prism	The foraging (swarm) robot scenario [101].
[115]	Process Algebra for Robotic Systems (PARS)	Search for a Biohazard – this seems to be for the verification of the underlying control system
[157]	Z model for self-adaptive systems	MAPE-K reference model [90] and Layered self-adaptive robotics software [53].
[108]	KeYmaera for verifying discrete and continuous behaviour, dynamic differential logic for hybrid systems	Robotic ground vehicle navigation
[155]	Gwendolen, SPIN, AJPF	Model checking for certification of autonomous pilotless aircraft systems.
[18]	BIP (Behaviour-Interaction-Priority) component framework. Verification in BIP using Evaluator model-checker (temporal logic properties) and monitors.	DALA (iRobot ATRV)
[34]	Theoretical work inspired by LTL and model-checking, computational framework where task specifications are regular expressions. Finite State Automata.	car-like robots in an urban-like environment
[122]	Use the synchronous language, Quartz, and then model-checking LTL/CTL specifications using Beryl model-checker in the Averest verification framework	Behaviour-based control network of mobile outdoor robot RAVON.
[36]	A metamodel based translation from AADL to BIP. Model-checking using Aldebaran (part of BIP) for deadlock detection and observers for monitoring safety properties	Flight computer.
[117]	Probabilistic verification using Monte Carlo approach (no tool support, theorems and proofs in paper).	Collision avoidance management of a large and changing number of autonomous vehicles (agents) in a shared environment.
[42]	SPIN to verify *Ach* (novel interprocess (IPC) communication mechanism and library).	Implementation of *Ach* on Golem Kang humanoid robot.
[83]	Timed Automata and Timed CTL (TCTL) to verify MAPE-K templates for the development of self adaptive systems.	There were 4 case studies carried out using the templates: (1) traffic jam monitoring system, (2) mobile system to support outdoor learning activities, (3) robotic transportation system and, (4) mobile digital storytelling application.
[112]	Hybrid system modelling using natural language programming (sEnglish) which is translated to Stateflow and verified using MCMAS.	Autonomous underwater vehicles.
[25]	(Extended Abstract) Brahms modelling framework translated into Jason agent code and model-checking. Temporal Logic.	Astronaut-robot collaboration.
[73]	Systems of differential equations and manoeuvrer automata generated using reachability analysis.	Collision avoidance of road vehicles.
[125]	LTL, extension to LTLMoP toolkit for robot mission planning.	Fire fighting scenario.
[21]	BIP framework	DALA autonomous rover
[130]	Extensible Agent Behaviour Specification Language (XABSL) - language for describing state machines.	Robot Soccer.
[123]	Synchronous language Quartz and verification using model-checking techniques of the Averest verification framework (Beryl)	Mobile outdoor robot RAVON
[81]	Timed Automata (TA) to model the system and Timed Computation Tree Logic (TCTL) for describing safety and correctness properties for an adaptive autonomous system	Fully distributed network of traffic monitoring cameras that reorganise to coordinate data transmission
[54]	RV-BIP, a tool that produces runtime monitors for robots the BIP framework from formal descriptions of the system and monitor.	autonomous rover
[82]	TA for modelling different modes of operation, and a module that enables them to be swapped during their execution on a Virtual Machine (VM)	Turtlebot robots navigating a map, representing a goods-transportation system.
[159]	Model the macroscopic behaviour of a swarm using a Probabilistic Finite-State Machine (PFSM), estimate the transition probabilities and then validate them using simulations in Player/Stage.	Swarm Navigating with the alpha algorithm.
[41]	Layered modelling of a robot's task plans using Petri Nets.	Robot Soccer
[158]	Reference model for building verifiable self-adaptive systems (FORMS).	Traffic monitoring system.
[33]	Modelling an abstract architecture for *reactive* multi-agent systems, which are then analysed for deadlock freedom.	Small example with two agents cooperative to move objects.
[162]	Modelling plans as Petri nets – with an additional set of Goal Markings – for single- and multi-robot systems. They capture dynamic, partially observable environments.	Multi-robot foraging and object passing.
[106]	Uses the process algebra Bio-PEPA to model the Marco and Micro level behaviour of robot swarms. These are analysed using the Bio-PEPA Tool Suite and PRISM.	Swarm foraging, where a team of three robots is needed to carry each object.
[93]	Uses model checking to find runs of a global system specification that satisfy an a version of Linear Time Logic without the 'next' operator (LTL-X) property, then uses that run – and Büchi Automata – to produce a robot control and communications strategy with reduced 'stop-and-wait' synchronisations.	A team of unicycle robots with distributed control.
[100]	Specifies robot behaviour in Z, then links Z animator Jaza to the Java debugger ***jdb*** to monitor the running program without alterations or additions.	Robot assembly system, for building robots for the NASA ANTS project [1]
[5]	Derives a specification of a real-time system (in CSP+T) from a UML-RT Model to enable design-time schedulability and dependability analysis.	Two-armed industrial robot.
[67]	Presents a methodology for verifying three cooperative robots with distributed control. The work uses the formal language Klaim, which has been designed for distributed systems, and its extension for stochastic analysis.	Three cooperative robots transporting an object to a goal area.
[10]	Ontology-based (OWL) reconfiguration agent in a manufacturing environment.	Manufacturing environment with four machines (processing machine, handling machine, Simple Conveyor Chain and filling station).
[146]	KnowRob ontology which is based on description logic, OWL	Household robot setting a table
[121]	IEEE-RAS Ontology for robotics and automation	
[89]	Generating robot motion plans that satisfy properties in a deterministic subset of a strict superset of other temporal logics (μ-calculus).	
[99]	Generating robot behaviour automata that satisfy Linear-time Temporal Logic (LTL) specifications, uses Linear Temporal Logic MissiOn Planning (LTLMoP) and TuLiP.	Driverless car in an urban environment.

Table 1: Results from search.

1.1 Methodology

In this section, we outline the methodology that was followed when conducting this review. First we discuss the factors that delimit the scope of our survey, then we identify the research questions that we are attempting to answer and outline the search criteria that were followed.

Scope: Our primary concern is the specification and verification of autonomous robotic systems. This hides a wealth of detail and often conflicting definitions. Therefore, we delimit the scope of our survey as follows:

- We target systems that (eventually) have some physical effect on the world, not purely software-based systems. In this sense, some of the work we examine falls within, though does not fully cover, the realm of *cyber-physical systems* (which are typically tackled by hybrid formalisms);
- We cover not only systems that affect people but also those that can be controlled by people, though we do not model human behaviour in detail as this is beyond the current capabilities of formal methods. Consequently, we do not formalise complex behavioural models from Psychology;
- While some robotic systems may, as above, be controlled remotely by a human, we allow for the full range of autonomy, from human controlled, to adaptive (i.e. driven by environmental interactions) and on to fully autonomous systems (i.e. that can choose to ignore environmental stimuli and make their own decisions).
- We address a range of formal properties, though primarily concerning the behaviour of autonomous robotic systems. In particular, we do not consider mechanical, materials, or physics issues. Broadly, we consider functional behaviour, such as safety, security, reliability, efficiency, etc;
- We limit the formalisms to those commonly encountered within the formal methods literature – logics, algebras, automata, formal models, etc. We do not cover techniques based on, for example, systems of differential equations except to mention them as part of hybrid approaches [24];

Research Questions: Our objectives in this paper are to enumerate and analyse the formal methods used for (autonomous) robotic systems. To this end, we seek to answer the following three research questions:

RQ1: What are the difficulties in formally specifying and verifying the behaviour of (autonomous) robotic systems?

RQ2: What is the current best practice concerning formalisms, tools, and approaches for addressing the answer to RQ1?

RQ3: What are the current limitations of the answers to RQ2 and are there developing solutions aiming to address them?

We answer RQ1 in §3 where we identify the current *external* challenges to formally specifying and verifying robotic systems. We add to this answer in §4 where we discuss the *internal* challenges posed when specifying and verifying various styles of autonomous robotic systems. We provide a thorough analysis of our findings in § 5 and 6, in which we examine potential answers to RQ2 and RQ3. There are, of course, many other questions that could be posed but we begin with the above. Indeed, these are enough to expose a wide variety of research in the core areas.

Search Criteria: We focussed on the *formal modelling*, *formal specification*, and *formal verification* of (autonomous) robotic systems. Our search travelled 5 levels deep of Google Scholar[2] and discounted any results that were obviously out of scope. In total, we surveyed 156 papers of which 41 were deemed to be in scope. We restricted this search to papers that have been published in the last ten years (2007 – 2018). In order to contextualise these results we did some 'snowballing' from the core set of papers that were discovered and these will be mentioned throughout this survey, however, our analysis specifically focuses on this core set and no others were included in order to preserve the integrity of our results. The relevant publications found during our search are summarised in Table 1; which describes the formalism(s) and tool(s) used, and any case studies or running examples described therein.

[2] Date accessed: 21/05/2018.

1.2 Related Work

We could find no existing survey of formal methods for robotics systems. This section discusses similar, related work and differentiates them from our work.

Whether deployed in hazardous situations or not, robots almost always have safety-critical aspects to be considered. A survey on safety-critical robotics [68] identified seven focus areas for the development of robots that can safely work alongside humans and other robots, in unstructured environments. These areas are: (1) modelling and simulation of the physical environment to enable better safety analysis, (2) formal verification of robot systems, (3) controllers that are correct-by-construction, (4) identification and monitoring of hazardous situations, (5) models of human-robot interaction, (6) online safety monitoring that adapts to the robot's context, and (7) certification evidence.

Nordmann et al. [114] present a detailed survey of Domain Specific Modelling Languages for robotics. While their study is influential to us in terms of its depth and organisation, it focusses on non-formal modelling languages. They analyse languages by both the applicable domain and development phase. The use of model-driven engineering approaches in building robotic systems seems too prevalent to ignore, as we discuss in §2.

Simmons et al. have identified three classes of verification problems for the automatic verification of autonomous systems. These are: (1) special purpose languages use special-purpose interpreters or compilers that need to be verified, (2) application-specific programs written in these languages needs to be verified for internal correctness (safety, liveness, etc.) and (3) these programs need to also be verified for external correctness (how they interact with the overall software system) [139].

Weyns et al. [156] present a survey of of formal methods for self-adaptive systems. While robotic systems may be required to adapt themselves, this behaviour is not required of all robotic systems. We discuss the specific challenges of formally specifying and verifying self-adaptive robotic systems in §4.3.

Rouff et al. [134, 133] survey formal methods for intelligent swarms. As we discuss in §4.2, the challenges posed by robot swarms are an extension of those of single autonomous robotic systems. Rouff et al. compare a wide range of formalisms, covering process algebras, model-based notations, various logics, and others. They come to the conclusion that no single formalism is suitable to the range of challenges posed by intelligent robot swarms; however, they focus more narrowly than this survey and the publication date of 2004 means that their work needs updating with a more recent survey.

Although our focus is on the application of formal methods, we recognise the important role that non-formal and semi-formal approaches have and continue to play in the development of reliable robotics. We summarise some of these briefly in §2.

2 Software Engineering for Robotic Systems

The complex task of engineering a reliable robotic system is often addressed by a variety software engineering techniques. Though the focus of this survey is formal approaches to specifying and verifying robotic systems, this section introduces some of the less formal approaches; particularly where they are potentially useful inputs to, or could be integrated with, a formal technique. We intend this section to provide the reader with a more rounded view of robotic system engineering. To this end, we have identified the following categories of non-formal and semi-formal methods:

Testing and Simulation: Analysis of robot system behaviour often relies on field tests [95, 61, 70, 111] (using the real robots) or simulations [95, 72, 8]. These approaches are undoubtedly useful for dealing with robot hardware and environmental testing, especially as we cannot always test robots in their intended target environment, due to safety, expense, or regulation. However, they are time-consuming, potentially dangerous to life and the robot hardware; and, since they only exercise particular sequences of program behaviours, they cannot be used to reason about properties of the program as a whole.

Middleware Architectures: Several component-based middleware architectures have emerged for programming robotic systems. Each adopts the paradigm of *components* (or *nodes*) and messages passing between them. These middlewares aim to simplify the engineering process and encourage reuse of existing software components. Among the most popular in the literature are: ROS [124], OPRoS [85], OpenRTM [12], Orocos [136], and G^{en}oM [60]. Each of these middleware frameworks tackles the problem of engineering a robotic system in a slightly different way, however they share a common set of component-based, modular concepts. To foster cooperation and reuse between these disparate frameworks, a definition of the abstract concepts that they implement is presented in [138]. The work identifies the component-based concepts shared by Orocos, ROS, G^{en}oM, and OpenRTM; though each architecture adopts these concepts in a slightly different way. Supporting the idea of integrating middleware frameworks, the work in [4] presents a robotic software manager with a high-level API that bridges the gaps between three of these different frameworks (ROS, OPRoS, and OpenRTM). Their top-down approach allows a single system to be engineered from components from multiple middleware frameworks. These two abstractions could be the basis of a formal technique for describing component-based systems. It is often assumed that that a middleware is sound, and this is key to trusting the robotic system [61]. However, given the heterogeneity of the systems that can be produced using such architectures and their parametrisable nature, guaranteeing correct robot behaviour can be challenging [70].

Domain Specific Languages: Another approach presented in the literature is to define Domain Specific Languages (DSLs) that describe specific sections of a robotic system. An extensive literature survey identified 137 state-of-the-art DSLs for robotics [114], the majority of which were for capturing robotics architectures and for use within more mature subdomains – such as robot motion. Conversely, fewer DSLs were aimed at emerging subdomains, such as force control. The majority of DSLs in the survey were designed to describe the basic components of the system at the early stages of development, but very few were suited to application at runtime. The identified DSLs were often defined using Ecore [144] meta-models, which could be a useful point of convergence to encourage the sharing of meta-models. This seems like a useful avenue of future work for integrating formal methods more closely with existing engineering practices. Related work on providing a robotics-specific Integrated Development Environment (IDE) [143] aims to tackle the problems of integrating the various different phases of robotics software development. They integrate a variety of DSLs into an IDE that guides the user through a standardised development process.

Graphical Notations: A variety of graphical notations are presented/examined throughout the literature. Usually, these aim to provide an easy to use way of designing robotic software that enables communication between engineers from the various distinct disciplines involved in robotics [71, 150, 107]. Statecharts [71] form the basis for several of these graphical notations. For example, ArmarX statecharts [150] provide a generic method for describing robotic systems. Each state has four different phases: *onEnter*, *running*, *onBreak*, and *onExit*. Individual phases are linked to a user-defined function enabling developers to run their own code during different phases of a state. Transitions between states facilitate the transfer of data and ArmarX Statecharts are executable, as C++ programs. A similar executable notation is *restricted Finite State Machine* (rFSM) Statecharts [107], which represent a subset of UML 2 and Statecharts [71]. Again, states have actions that occur at different phases of the state: *entry*, *exit*, and the in-state activity *do*. States perform an action in response to an event. Transitions between states can have a boolean guard and an action that occurs during the transition. Other notations include RoboFlow [9] which adopts the style of flow charts, boxes represent procedures and directed arrows between them indicate data flow. Diamonds represent a choice, with one input and multiple outputs. No control flow constructs (such as *while* or *repeat*) are present in this notation. RoboFlow is specifically designed for engineering robot movement and manipulation tasks.

MDE/XML: A number of Model-Driven Engineering (MDE) approaches use the Eclipse Modelling Framework's Ecore meta-meta-model [59]. The work in [79] presents a mapping from the XML-based AutomationML into ROS program code. The BRICS Component Model [32] enables high-level models to be translated into platform-specific component-models, specifically targeting Orocos and ROS. Another model-driven approach designed for component-models is the MontiArcAutomaton architecture description language [129]. This approach can also translate models into program code, mostly focussing on the ROS framework [2]. Performance Level Profiles [28] is a novel XML-based language for describing the expected properties of functional modules. They provide tools for the automatic generation of code for runtime monitoring of described properties. They again target the ROS framework, but we note that the output from their tool is flexible and not tied to ROS. RADAR is a novel procedural language for describing event responses for robots that is implemented in Python [23]. The Declarative Robot Safety language (DeRoS) [3] is a DSL for describing a robot's safety rules and the corresponding corrective action the robot should take. Usefully, this provides automatic code generation to integrate these rules with runtime monitoring of a ROS program by generating a ROS safety monitoring node.

These non-formal and semi-formal approaches aim to adapt the parts of software engineering practice that are relevant to the robotics domain. Middleware frameworks provide a flexible standard interface to the robotic hardware. DSLs can be used to describe properties about specific parts of the system in an abstract way, often improving the understanding or interoperability of the system. Graphical notations aid in the communication and visualisation of ideas. However, more formal languages provide unambiguous semantics and enable reasoning about the whole program rather than only a particular set of program behaviours. In the next section, we describe the *external* challenges faced when applying formal methods to robotic systems.

3 External Challenges of Autonomous Robotic Systems

In this section, we discuss the external challenges that are encountered when specifying and verifying robotic systems. We label these as *external* challenges because they are independent of the underlying robotic system being developed and will be encountered by all such systems. Firstly, we discuss the challenge of modelling the physical environment of a robotic system, which is of paramount importance since real-world interactions can significantly impact safety. Approaches to modelling the environment often use non-formal simulations [95, 72, 8] based on differential equation descriptions of Physics, Chemistry, Mechanics, etc. This aspect is a challenge for formal and non-formal approaches alike, due to the continuous dynamics of a robot's environment. The physical environment has an effect on all robotic systems and we specifically discuss this challenge in § 3.1.

Secondly, we discuss the challenge of providing sufficient 'evidence' for certification. Often, robotic systems require certification because they are deployed safety-critical situations; for example, the nuclear industry. Other robotic systems operate in areas that are weakly regulated but require public trust; for example, domestic assistants. Some robotic systems require both certification and public trust; for example, autonomous vehicles. The challenges of enabling the formal verification of robotic systems in a way that can be used as safety certification evidence and to build public trust are specifically discussed in § 3.2.

3.1 Modelling the Physical Environment

Broadly, we consider a robot as a machine that implements Artificial Intelligence (AI) techniques and has a physical presence in the world. Given this physical interaction, one of the most prominent challenges in verifying robotic systems involves its interaction with an unstructured environment. This is further complicated by differing sensor accuracy or motor performance, or components that are degraded or damaged. In fact, differing performance of sensors can be an issue even without any degradation. We note that although several authors consider tele-operated entities as robots, we focus particularly on those that can exhibit more autonomous behaviour.

Interactions between a robot and its physical environment can have a major influence on the robot's behaviour, because the robot may be reacting to unforeseen environmental conditions. Indeed, adaptive systems exhibit behaviour directly driven by their environmental interactions. While it is accepted that formally modelling a robotic system *within* its physical environment is important [151, 58], it has had limited attention in the literature. We believe that this is due to the sheer complexity of this task and even if an accurate real world model could be achieved, the robot only has a partial knowledge of its surroundings due to sensing limitations caused by sensor blindspots and interference between sensors [26, 118, 73].

Some techniques focus solely on the robotic control software, ignoring the environment [102, 95]. Others assume that the environment is static and known, prior to the robot's deployment [111, 153, 63], which is often neither possible nor feasible [73]. For example, the environment may contain both fixed and mobile objects whose future behaviour is unknown [26] or the robot's goal may be to map the environment, so the layout is not known. Hybrid architectures often take a third approach; to abstract away from the environment and assume that a component can to provide predicates that represent the continuous sensor data about the robot's environment [147, 116, 58, 48, 45]. While this encapsulates the high-level control in such a way as to insulate it from the environment, the assumption of a static known environment can sometimes be implicit.

Using models of the environment that are static and assume prior knowledge raises questions about the validity of any results following from its use. In one promising study, Sotiropoulos et al. [142] examined the ability of low-fidelity environmental simulations to reproduce the bugs that were found during field tests. They took the bug logs from the field tests of a robot and tried to reproduce them in a simulation of the same robot and its environment. They found that of the 33 bugs that occurred during the field test, only one could not be reproduced in the low-fidelity simulation. Perhaps similar results might be obtained with low-fidelity formal models of a robot's environment.

One approach to modelling a dynamic and uncertain environment has been to capture it in a probabilistic model [113]. Here, a PRISM model captures the environment of a domestic assistant robot. Non-robot actors in the environment are specified using probabilistic behaviours to represent the uncertainty about their behaviour. The robot model is also written in PRISM, so that it can be reasoned about within this probabilistic environment. This is a useful step, that accepts the changing nature of the environment. However, for the model-checker to explore outcomes based on a certain behaviour, its probability must be encoded into the environment model. This still leaves the possibility of unforeseen situations having a detrimental effect on the robot's behaviour.

Hoffmann et al. [78] use a similar probabilistic approach, formalising a pilotless aircraft and its physical environment using an extension of PFSMs. They use PRISM to verify properties about their model, with the pilotless aircraft tasked with foraging the environment for objects, which it must return to a drop-off location. They take an interesting modelling approach, running small-scale simulations of the pilotless aircraft in its environment to determine the probability and timing values for their formal model.

An alternative approach is taken by Costelha and Lima [41], who divide their models into layers and try to obtain a model of the robot's environment that is as realistic as possible. Their work uses Petri Nets, and whereas other layers in the models use stochastic Petri nets, the environment model uses untimed direct transitions.

To address the challenge of combining discrete computations and a continuous environment, a robotic system is typically separated into several layers [7, 58]. At the bottom, the functional layer consists of control software for the robot's hardware. Then, the intermediate layer generally utilises a middleware framework (such as ROS [124] or G^{en}oM [60]) that provides an interface to the hardware components. The upper layer contains the decision making components of the robotic system, which capture its autonomous behaviour.

Modelling the environment is of particular relevance when attempting to navigate through it and, although a number of navigation algorithms exist, not all can be confidently employed in safety-critical scenarios because few have been verified [118]. The Simplex architecture [137] provides a gateway to these uncertified algorithms as advanced controllers (AC). Here, they are used alongside a pre-certified baseline controller (BC) and should anything go wrong with the AC, the BC will take control. The decision module in the Simplex architecture and a switching logic allow a robotic system to change between algorithms at runtime to avoid collisions, with only limited information about obstacles [118]. Other work includes [73] which employs reachability analysis to generate a manoeuvrer automaton for collision avoidance of road vehicles. Their work models the continuous behaviour of the system using differential equations. Fault monitoring, such as these runtime verification techniques, is useful for catching irregular behaviours in running systems though still does not solve the problem of ensuring safety in all situations.

There has been much work done on collision avoidance and safe navigation of robots [118, 26, 108, 122]. Mitsch et al. [108] use differential dynamic logic (dL) [119], which was designed for hybrid systems, to describe the discrete and continuous navigation behaviour of a ground robot. Their approach uses hybrid programs for modelling a robot that follows the dynamic window algorithm and for modelling the behaviour of moving objects in the environment. Using the hybrid theorem prover KeYmaera, Mitsch et al. verified that the robot would not collide with stationary or moving obstacles, and maintains sufficient distance from obstacles. They report that, in proving these safety properties, 85% of the proof steps being carried out automatically. In an update of [108], Mitsch et al. [109] verify a safety property that makes less conservative driving assumptions, allowing for imperfect sensors; and add liveness proofs, to guarantee progress. Further, they extend their approach to add runtime monitors that can be automatically synthesised from their hybrid models. They note that all models deviate from reality, so their runtime monitors complement the offline proofs by checking for mismatches between the verified model and the real world.

Formal and non-formal models of the real world are prone to the problem of the reality gap, where models produced are never close enough to the real world to ensure successful transfer of their results [151, 58]. This is particularly problematic when real-world interactions can impact safety. Bridging the gap between discrete models of behaviour and the continuous nature of the real-world in a way that allows strong verification can often be intractable [49]. There is also a trade-off between ensuring that the system is safe and ensuring that it is not so restrictive that it is unusable in practice [151].

3.2 Trust and Certification Evidence

Robotic systems are frequently deployed in safety-critical domains, for example nuclear or aerospace. These systems require certification, and each domain usually has a specific certification body. Ensuring that the engineering techniques used for developing robotic systems are able to provide appropriate certification evidence is, therefore, essential. However, autonomous robotic systems development often falls short of providing sufficient evidence, or evidence in the correct form, for these certification bodies [61].

When certifying a piloted aircraft, the certification bodies assume that it will have a suitably qualified and competent crew in control. When an autonomous system is in control of a pilotless aircraft, this system must also be certified. However, the regulatory requirements of such a pilotless aircraft are still under development by the relevant regulatory bodies [152]. Choosing an appropriate formal method for a particular system is no mean feat as there are currently no standard guidelines available to aid developers in choosing the most suitable approach [96]. This also presents a challenge for certification bodies who need to be able to determine the reliability of safety-critical robot system.

In contrast to the nuclear and aerospace domains, robotic systems such as household assistants do not yet have strong certification bodies controlling their safe use. In the UK this aspect falls to either the local authority in which the system is deployed or, sometimes, the national Health and Safety Executive. In this context, arguably more so than with autonomous vehicles, robots must be shown to be safe and trustworthy. This covers both safety and the public perception of safety; notions of usability and reliability; and a perception that the robot will not do anything unexpected, unpleasant or unfriendly [51]. This lack of both trust and safety assurances can hamper adoption of robotic systems in wider society [63], even where they could be extremely useful.

A topic related to trust is the issue of ethical robotic behaviour. Several approaches to constraining the behaviour of a robotic or autonomous system have focussed on the distinction between ethical and non-ethical choices [15, 16, 31]. Worryingly, they ignore the potential situation of there being no ethical choice available, and there seems to be little consideration of this situation in the literature [45]. Clearly,checking ethical behaviour is as difficult with robots as it is with humans.

Formal methods may also be applied to this domain as can be seen in Rizaldi et al. where Isabelle/HOL and temporal logic were used to formalise a subset of traffic rules for vehicle overtaking in Germany [131]. However, as we move towards driverless cars a key concern is enumerating how much one robot needs to know about the goals and constraints of other vehicles on the road to not be considered to blame [108]. In related work, Webster et al. utilise model checking to provide certification evidence for an autonomous pilotless aircraft systems using the Gwendolen agent language and the Agent Java Path Finder (AJPF) approach as an improvement over the Promela and the SPIN model checker [155].

The assembly of safety cases can be used to provide evidence for certification bodies. A safety case is a structured argument that is supported by a collection of evidence providing a compelling, comprehensible and valid case that a system is safe for a given application in a given environment [43]. Automating the generation of such documentation is a challenging task as they need to account for heterogeneous content such as physical formulae from the design of the physical system, maintenance procedures and software (which itself, may be of a heterogeneous nature). Denney and Pai [43] have described a methodology for automatically generating such safety cases for the Swift pilotless aircraft system using a domain theory and their AutoCert tool [44], which automates the generation of safety case fragments.

3.3 Summary of External Challenges

In this section, we identified two primary external challenges to the specification and verification of robotic systems. These are (1) modelling the physical environment and (2) providing sufficient (and appropriate) trust and certification evidence. With respect to modelling the physical environment, two approaches appear to dominate the literature, these are to either model or monitor the physical environment. Temporal logics, for example, have been used to model the environment with model-checking used for verification. Specifying a monitor to restrict the robotic system to safe behaviours within its environment reduces the verification burden, as only the runtime monitor needs to be verified. However, comprehensively capturing *all* unsafe situations so that they can be mitigated in advance is extremely difficult.

Certain tasks, such as navigating an unknown and dynamic environment, are challenging for robotic systems; and a number of navigation algorithms exist in this domain. However, not all can be employed in safety-critical scenarios as they have not been verified [118]. This suggests that there are limitations in the use of current formal methods to verify these algorithms, and leads to hybrid approaches that have high computational complexity. Clearly, a recurring challenge is the complexity of formal models, particularly if we consider the modelling of a complex physical environment.

Progress has been made in enabling formal techniques to provide trust and certification evidence, but it is clear that current techniques are not sufficient. This is further complicated by the fact that there are currently few guidelines to help developers identify the most appropriate formal method to specify and verify their system [96]. Although there has been recent progress, regulators and certification bodies tend to be hesitant to suggest suitable formal methods for safety-critical systems. Therefore, regulators, developers and academia alike face the challenge of how to determine suitable and robust formal methods for different types of robotic systems. The *internal* challenges, stemming from how a robotic system is engineered, are discussed in §4.

4 Internal Challenges of Autonomous Robotic Systems

Autonomous robotic systems are becoming more prevalent and are extremely attractive with respect to removing humans from hazardous environments such as nuclear waste disposal or space exploration. There are a variety of ways to design these systems using artificial intelligence approaches such as agent-based systems which we discuss from the perspective of formal specification and verification in §4.1. §4.2 and §4.2.1 describe the difficulties encountered when formally specifying and verifying homogeneous and, respectively, heterogeneous multi-robot systems. Moreover, when such are deployed into an environment that cannot be easily or safely accessed by humans they must be fault tolerant and be capable of self-adaptation and reconfigurability should a system or hardware failure occur. In §4.3, we discuss some approaches to specifying and verifying self-adaptive and reconfigurable systems.

4.1 Agent-Based Systems

The Beliefs Desires Intentions (BDI) model is dominant in the theoretical foundations of rational agency [64]. Based on the BDI model, the Procedural Reasoning System (PRS) was developed as an agent architecture [65]. This led to a vast array of work [126] resulting in this approach becoming the predominant model of autonomy in the literature. Agents can appear both individually and in multiples depending on the system being developed.

The distributed Multi-Agent Reasoning System (dMARS) has been constructed as an implementation of PRS with the BDI model operationalised by plans in dMARS [50]. A formal specification of dMARS has been developed in Z that provides a formal foundation for the semantics of dMARS agents [50]. These agents monitor both the world and their own internal state. They are comprised predominantly of a plan library, functions for intention, event and plan selection, belief domain and goal domain [50].

Webster et al. [155, 152] build a BDI agent program that controls an autonomous pilotless aircraft, using the BDI agent language GWENDOLEN [46]. Their focus is on verifying the agent, which is taking the role of the pilot, against the rules and recommendations that civilian pilots must follow in UK airspace (as specified by the Civil Aviation Authority). The verification is achieved using the AJPF model-checker [47] and the verified agent is then plugged into a flight simulator [152] for a visualisation of the verified scenarios. However, since AJPF is a program model-checker, it is substantially slower than traditional model-checking techniques. They report that comparing SPIN to AJPF, the time required to verify one property increased from milliseconds to minutes or hours [155].A similar BDI agent, also programmed in GWENDOLEN is used in [86, 87], where the agent is controlling a driverless car in a vehicle platoon. They use AJPF to verify safety properties for the car joining and leaving the platoon, and maintaining a safe distance curing platooning. Verification of the timing properties is handled by Uppaal.

Using a similar program model checking approach, Choi et al. model a system of nine heterogeneous agents with one representing the robot's physical environment [37]. They used the Model Checker for Multi-Agent Systems (MCMAS) to verify the system and illustrated that MCMAS performed dramatically better than SMV in this setting. Molnar and Veres also used MCMAS to verify autonomous underwater vehicles [112]. They use natural language programming (sEnglish) which results in an ontology that is used to represent a labelled transition system that is then translated to Stateflow and verified with MCMAS. Interestingly, in this work they represent the human interface as an agent, as does [25]. In this way they provide a link between the human behaviour and the formal model that they have developed.

In [78], Hoffmann et al. characterise the autonomous behaviour of an agent (loosely based on the BDI model) as an PFSM extended with a weight function to map weights onto actions, which they call an *Autonomous* Probabilistic Finite-State Machine (APFSM). Because of the addition of the weight function, an APFSM is a discrete-time Markov chain. They use this formalism of agent autonomy to verify properties about a pilotless aircraft on a foraging mission.

Bozzano et al. proposed their Autonomous Reasoning Engine (ARE) that is to be embedded into a spacecraft. It interacts with the communication centre on the ground, provides autonomous reasoning capabilities, receives information from sensors and sends commands to actuators [27]. A prototype of ARE has been implemented based on the NuSMV symbolic model checker and it has been used in ESA rover and satellite projects providing a model-based approach to on-board autonomy. ARE supports plan generation, validation, execution, monitoring and Fault Detection, Isolation and Recovery (FDIR).

Podorozhny et al [120] use Alloy to verify multi-agent negotiations, their work provides a mechanism for checking coordination and interaction properties of multi-agent systems, as well as properties of complex data structures that the agents may be manipulating or sharing. Their work centres on a system where multiple agents strive together to reach some common goal, in such a system agents must negotiate to optimise achieving their common goal(s). They have modelled the negotiations that the agents must complete as Finite-State Machines (FSMs), these are then translated into an Alloy specification that can be verified using Alloy Analyzer. The main difficulty that was encountered was ensuring that the constructed Alloy model remains under tractable scope while checking its properties.

4.2 Homogeneous Multi-Robot Systems: Swarms

Historically, the development of multi-robot systems has taken inspiration from biological systems such as swarms of insects. A robot swarm is a decentralised collection of robots that work together [20]. The robots in a robotic swarm system generally exhibit five key criteria, they are: adaptive, present in large numbers, not in homogeneous groups, relatively incapable or inefficient, and capable of local sensing and communication [135]. However, these criteria are not exhaustive, and should simply be used to decide to what degree the term "swarm robotics" might apply [52].

Swarm (and, more generally, multi-agent) robot systems can present difficulties for ensuring safety and liveness properties due to the number of concurrent, interacting agents and the changeability of the system's environment [6]. Formal methods have obvious benefits for specifying the behaviour of swarm robot systems, because field test and simulations only cover a particular instance of the swarm's behaviour [95]. However, the large number of robots in a swarm can cause state space explosion and hinder the verification of swarm behaviour using conventional model-checking approaches [95]. A further issue is ensuring that if you prove a property for a swarm of n robots, that the same property holds for different sized populations..

Swarm robotic systems exhibit some similarities with the "Systems of Systems" concept, which presents another challenge for the application of formal methods. This is also present both in wide-ranging work on hierarchical multi-agent systems and organisations, as well as hierarchical systems. Specifications of the behavioural requirements of a swarm can be at the microscopic level – that is, individual robots – or at the macroscopic level – that is, of the whole swarm. Formally relating the formalisms used at the macro- and microscopic levels can ease the development of robot swarms that implement behavioural requirements at the microscopic level, that are specified at the macroscopic level. Smith and Li [141] present MAZE, an extension of Object-Z for multi-agent systems. Their approach employs Back's action refinement to facilitate the development process from macro to micro level and they devise a number of syntactic conventions to support micro-level specification.

Robot swarms are often used for reasons of resilience. Winfield and Nembrini [160] examined fault tolerance in robot swarms navigating using the alpha algorithm and found that robustness arises from the inherent properties of a swarm: simple parallel robots, operating in a fully distributed way, with high redundancy. Of particular interest, they found that a robot's *partial* failure caused the swarm more problems. In their example, a robot that was fully functioning apart from its wheels anchored the swarm in place. Other challenges include determining if such failures will propagate through the swarm, and determining a lower bound on the number of robots that must remain functioning for the swarm to complete its mission [98]. Kouvaros and Lomuscio [98] present an approach that uses a temporal logic and fault injection to determine the ratio of faulty to functioning robots in a swarm. Their case study swarm also navigates using the alpha algorithm. They specify temporal-epistemic properties and then find the threshold at which the swarm no longer satisfies these properties. Crucially, their approach is designed to work with swarms where the number of agents is unknown at design-time.

Programs for robot swarms are often developed in an ad-hoc manner, using trial and error. They tend to follow a bottom-up approach that involves programming individual robots before combining them to form a swarm which displays a certain set of behaviours [30, 102, 111]. This development process can reduce the effectiveness of formal engineering approaches; the application of formal methods often does not guarantee that the implementation matches the specification, because the programs are built manually from the specification [102]. Winfield et al. [159] present the straightforward approach of modelling the macroscopic behaviour of the swarm using a PFSM. First, they estimate the transition probabilities in the PFSM; then, they validate these probabilities by simulations in Player/Stage.

Rouff et al. compared four different specification formalisms (Communicating Sequential Processes (CSP), Weighted Synchronous CCS (WSCCS), Unity Logic, and X-Machines) for specifying and verifying emergent swarm behaviour [134]. Their conclusion was that, a blending of these formalisms offered the best approach to specify emergent swarm behaviour as none was sufficient in isolation. It is thus clear that only the use of multiple, specialised tools and methodologies can achieve a high level of confidence in software [77]. For example, the NASA Remote Agent uses specialized languages for each of the planner, executive and fault diagnosis components [139]. We explore this approach to the specification and verification of robotic systems in §6.4.

The work in [94] identifies a cultural resistance to adopting formal methods in developing robot swarms (which presumably extends to other domains as well). There is a perception that applying formal methods is a complicated additional step in the engineering process, one which prolongs the development process while not adding to the value of the final product. A lack of appropriate tools also often impedes the application of formal methods. Since this work was cited in [102] (published in 2016, nineteen years after [94]) we assume that these challenges are still present.

The work in [67] presents a methodology for verifying cooperative robots with distributed control. They use the formal language Klaim, which has been designed for distributed systems; its stochastic extension, StoKlaim; and its related tool set. Their methodology involves building a specification of the robot's behaviour and their physical environment, in Klaim; and then adding stochastic execution times for actions, in StoKlaim. They validate their modelling approach by comparing their results to those obtains from physics-based simulations of the same system. They describe this approach using a case study with three homogeneous robots, cooperating under distributed control, to transport objects. This collection is too small to be considered a swarm, but the behaviour of each robot is the same. They intend to scale this approach up to larger collections, more properly swarms.

4.2.1 Heterogeneous Multi-Robot Systems: Teams

Not all robotic systems are structured as swarms; robotic systems may consist of a team of heterogeneous robots that must work together in order to achieve a specific (set of) goal(s). In hazardous environments such as nuclear plants, it is conceivable that multiple distinct robots may be required to complete a specific task. In this setting, each robot would have a specific and independent role, for example, a robotic arm to examine a piece of nuclear waste and a more mobile robot to monitor, calibrate and mend the robotic arm should it malfunction. The robot team can be easily specified and verified at the macroscopic level, but at the microscopic level each robot must be specified and verified individually. For example, one robot in the team may be characterised by a verifiably correct Z specification, whereas another may be model-checked at the source code level. The behaviour of the heterogeneous robot team could be different to the behaviour of each individual robot and it might not be the straightforward composition of the behaviours of each robot. Therefore, another formalism may be used to specify and verify the team's behaviour. Linking the specification and verification approaches at these various levels of abstraction is difficult because the approaches most amenable to each might be different.

In an attempt to automate the development of robot teams, the work in [93] presents a methodology that model checks a transition system describing the behaviour of the robot team, for execution traces that satisfy an LTL-X formula. This trace is then mapped to the communication and control strategy for each robot. Additionally, the work uses Büchi Automata to produce a communication strategy that reduces the number of 'stop-and-wait' synchronisations that the team has to perform.

4.3 Self-Adaptive and Reconfigurable Systems

Ensuring fault tolerance of robotic systems that are operating in hazardous environments, potentially far away from any possible human interaction, is crucial. These systems must be able to diagnose and reconfigure themselves (and possibly each other) in order to adapt to changes in their requirements or operating environment. Reconfiguration can help maintain quality of service or meet functional objectives [157]. This is particularly relevant with respect to robotic systems being deployed in an environment that is hostile to humans, for example in space or the deep ocean [145].

A self-adaptive system continually alters its behaviour in a feedback loop that is driven by its environment. The findings of a literature survey indicated that there are no standard tools for the formal modelling and verification of self-adaptive systems with 30% of the tools surveyed using model-checking [156].

Architecture-based self-adaptive systems constitute a specific class of self-adaptive system where the system reasons about a model of itself and the environment. It then adapts according to some adaptation goals. In this case, the feedback loop consists of Monitor, Analyse, Plan and Execute components and runtime models which provide Knowledge. This structure is called MAPE-K [83]. A series of MAPE-K templates has been formally verified using timed automata and TCTL to aid in development.

The development of reconfigurable machines is an area of the literature where a substantial amount of research has been carried out with respect to the hardware for such machines, but developing software that can autonomously reconfigure the system is a challenge for the entire community [22]. The development of such systems places a heavy emphasis on designing these systems to be modular thus making reconfiguration a more approachable endeavour.

One approach to providing reconfigurability is to develop a flexible control system that can reconfigure itself when a fault is detected [29]. The work in [82] presents a collection of TA models for different modes of operation, and a module that enables them to be swapped during their execution on a VM. Executing the TA models ensures that the model's verified behaviour is the program's runtime behaviour.

One avenue of research suggests providing (both semi-formal and formal) models to be used at runtime [35]. This agenda considers approaches such as automatic test case generation and formal model-checking. This relies on the fact that many variables that are unknown during the design of a system can be quantified at runtime, which helps to control the problems of state space explosion. This approach has the benefit of providing online assurance of an adaptive system, *while* it is adapting; which can improve the trust in results regarding safety.

Related to the notion of adaptation is the concept of a reconfigurable system, which senses the environment and makes a *decision* about how best to reconfigure itself to suit changes in its requirements or the physical environment. Reconfiguration is essential for ensuring the fault-tolerance of a safety-critical robotic system [145]. There seems to be two key research questions for formal methods that can be applied to reconfigurable systems: (1) how can a reconfigurable system be specified and analysed [113], and (2) how can two configurations of the system be compared [157, 113]?

Decentralised systems can bring their own set of challenges for adaptation and reconfigurability. Iftikhar and Weyns [81] use a traffic monitoring system as their case study, where a collection of cameras monitor the traffic on a stretch of road. When a traffic jam is detected, the cameras collaborate in a master-slave architecture to report information about the traffic. This self-adaptation is managed without a central control unit, to avoid the communications bottleneck this would cause. Iftikhar and Weyns model this system using TA and describe invariants and correctness properties using TCTL.

Since reconfigurability requires the system to make an autonomous decision about how to most suitably reconfigure itself, it is crucial that the decisions made by the system are *rational*, meaning that the system can

Formalism	References	Total
Temporal Logics	[95], [153], [84], [154], [122], [83], [112], [25], [81], [82], [125], [89], [99]	13
Discrete Event Systems	[83], [34], [81], [82], [159], [73], [125], [21], [130], [41], [33], [162]	12
Model-Oriented Specification	[157], [158]	2
Process Algebra	[115], [106], [5]	3
Ontologies	[112], [10], [146], [121], [69]	5
Other Formalisms	[58], [108], [112], [54], [130], [34], [67]	7

Table 2: Summary of the types of formalisms found in the literature search. Note: some techniques use more than one formalism.

explain, and justify its reasoning. This leads us to model the motivations and decisions of the system, ideally as first class objects that can then be assessed through formal verification[58].

4.4 Summary of Internal Challenges

In this section, we described the specification and verification processes that have been used to mediate the difficulties posed by the internal challenges of robotic systems – that is the different ways of engineering a robotic system. We discussed agent-based, multi-robot, adaptive and reconfigurable robotic systems from the perspectives of formal specification and verification. It is clear that the notion of a *rational* agent can help to mediate complexity in all of these scenarios. Of course, agent-based systems are one way of describing autonomy and there are many different models of agent-based systems that are based on different models of autonomy. Agents are used to model a robot's interactions with other actors, its environment, and the physical environment itself. The model of an agent can be used for both design- and run-time verification.

4.4.1 Answering RQ1:

RQ1 queried the difficulties encountered when formally specifying and verifying (autonomous) robotic systems. We have partitioned these challenges into two sets; those that are *external* (§3) and those that are *internal* (§4). The external challenges are those that arise when trying to deploy robots in the real world. The first external challenge that we identified was the difficult task of formally specifying and verifying the behaviour of the robot's physical environment. Secondly, the issue of providing sufficient evidence for public trust and certification is an avenue where current approaches fall short.

The internal challenges refer to the difficulties in formally specifying and verifying the robotic software system itself. This process is complicated by the internal structure of the software. To this end, agent-based systems, homogeneous multi-robot systems, heterogeneous multi-robot systems and issues related to self-adaptation and reconfigurability dominate our analysis.

5 Formalisms for Robotic Systems

The previous sections identified the challenges of robotic systems for formal methods; this section discusses the formalisms found in the literature search for specifying and verifying robotic systems. Table 2 summarises the categories of formalisms found; the most numerous category of formalism is temporal logics, though this covers a wide variety of temporal logics. The 'Other Formalisms' category contains those which do not fit in the other categories but are not numerous enough to warrant their own. Examples include formalisms that cater for the hybrid nature of robotic systems [108] and general specification languages for robotics [49].

Table 3 summarises the wide variety of formal analysis tools were found in the literature search. Several studies use tools that specify properties in temporal logic; for example, Uppaal [13], PRISM [30], and NuSMV [51]. The number of tools in use for temporal logic is commensurate with their prevalence in the literature (as shown in Table 2).

The wide variety of tools for the same formal approach points to a problem: the lack of interoperability between formalisms. Often, models or specifications accomplishing similar things are incompatible and locked into a particular tool. This suggests that a common framework for translating between, relating, or integrating different formalisms, would prove useful. Such a framework would ease or remove the work required to safely convert between formalisms or tools, and serve a growing need to capture the behaviour of a complex system using a heterogeneous set of models that each focus on a particular type of property.

Tool	References	Total
Prism	[95], [106]	2
Uppaal	[81], [82]	2
SPIN	[153], [154], [42]	4
Beryl	[122]	1
Aldebaran	[36]	1
Dfinder	[21]	1
AJPF	[58], [45]	2
MCMAS	[98], [37], [112]	3
KeyMaera	[108]	1
Bio-PEPA Tool Suite	[106]	1
TmeNET	[41]	1
TuLiP	[99]	1
LTLMoP	[99]	1

Table 3: Summary of the formal analysis tools used in the literature search. Note that a number of the contributions in the literature utilise multiple tools in order to achieve their goals. Sometimes this is due to the authors carrying out a comparison between tools and/or approaches. However, it is often the case that neither tool in isolation is capable of verifying the desired properties of the system at hand.

The use of intervals as a means for accommodating imprecision and techniques from computational geometry have proved useful [151]. Modular verification greatly simplifies the burden of the verification task, particularly when humans are required to interact with the provers [151].

In the remainder of this section we describe in detail the formalisms used to specify and verify robotic systems. First, we describe the use of existing formalisms: §5.1 discusses the various types of temporal logic that have been used, §5.2 describes approaches using automata, §5.3 describes approaches using process algebras, §5.4 describes model-oriented specification, and §5.6 describes other existing formalisms. Finally, §5.7 summarises the formalisms found in the literature.

5.1 Temporal Logic

Temporal logics are used for specifying properties about a system that have a dynamic component. There are a variety of temporal logics – such as LTL and Computation Tree Logic (CTL) – so it is no surprise that the formal verification of robotic systems makes great use of them and they feature heavily in the literature, as can be seen in Table 2. Temporal logics have been used extensively to address the challenges to developing reliable autonomous robotic systems that we described in the previous sections.

The work in [84] presents a probabilistic approach to modelling a robot, controlled by a BDI agent, and its environment. The agent can be captured as either a Discrete-Time Markov Chain or a Markov Decision Process. They then use Probabilistic Computation Tree Logic (PCTL) to specify the properties to be checked, using PRISM. They apply this technique to an autonomous mine detector, showing how their approach can be used for both design-time and runtime checking.

The work in [51, 63] presents the automatic modelling of the safety rules and environment of a robotic domestic assistant (a Care-O-Bot) in Probabilistic Temporal Logic (PTL). The Care-O-Bot's environment – a sensor-equipped UK domestic house – is modelled; the human isn't explicitly captured, but the house's sensors are able to change arbitrarily. Using NuSMV, these models can be checked for temporal properties and to ensure that the Care-O-Bot's rules are sufficient the keep the human safe. In [154], Webster et al. convert the rules of the Care-O-Bot, a human, and the same sensor-equipped house into Brahms, a multi-agent language. These rules were then automatically translated, using the *BrahmsToPromela* tool, into PROMELA. The Care-O-Bot was verified against its rules, within the model of the environment with a human, using SPIN. They used four different models of the environment, ranging from a model built from logs of a real human spending time in the sensor-equipped house, to a nondeterministic model where the modelled human could perform several actions at once. Webster et al. were able to verify the Care-O-Bot rules using each version of the environment model, giving them confidence in the safety of the Care-O-Bot's high-level decision making.

In [86], Kamali et al. verify the timing properties of a driverless car, using Uppaal. They use a combination of a timed temporal logic and a logic to describe the beliefs, desires, and intentions of a BDI agent. Crucially, they keep the two logics separate, to simplify the verification. To verify the timing properties of the whole system, the agent program is represented by an untimed automata. Separately, the BDI agent program is checked (using AJPF) against a untimed over-approximation of the environment.

Model-checking a system for safety rules is useful when trying to provide evidence for the general public perception of safety, without the help of any certification body. As mentioned earlier, Webster et al. [155, 152] use this approach to provide evidence for system certification by formalising the rules of the air using LTL with belief extensions. They also capture assumptions of safe and sensible pilot behaviour. This allows their models to be used to verify that an autonomous system controlling a pilotless aircraft follows the rules of the air in the same way as a pilot, providing useful certification evidence.

Several techniques enable the generation of behavioural models that satisfy a given temporal logic specification. For example,the technique presented in [89] can be used to produce motion plans that satisfy properties, specified in a deterministic subset of μ-calculus. Their approach incrementally builds a finite transition system, adding a sample of the possible steps that would move the robot towards its goal state until the transition system reaches the goal state and satisfies the specification; this transition system is then used as the motion plan. They present tests of their approach that produce a plan in ∼3.5 seconds for a system of over 1000 states; they suggest that their technique could be fast enough to provide a runtime planning facility. Another example [99] synthesises automata that describe the robot's behaviour, from LTL specifications, so that they behaviour satisfies the specification. They tackle the problem of state explosion by generating plans only a short distance ahead, replanning from their new location after they move.

Due to the parametrisability and heterogeneity of robot middleware architectures, temporal logics have been used to verify the behaviour of robot programs written using particular middlewares. For example, an approach targetting the G^{en}oM framework, which uses its existing tool chain to generate Fiacre models of the implemented system and then allows model checking against LTL properties [61]. Another example is the work presented in [70], which models ROS nodes and communications between them using Timed Automata and then uses UPPAAL to check them for TCTL. They enable the formalisation of concrete ROS applications, with variable architectural parameters. In their case study they verify the lack of message buffer overflows for a selection of queue sizes in the open-source robotic application Kobuki.

Through the close link with runtime verification, temporal logics have been used to develop models of monitors, which separate the specifications of the system's potential behaviour and its intended behaviour. The work in [105], aimed at autonomous systems in general, specifies a separate safety monitor using CTL. This safety monitor has independent access to the system's hardware and restricts the behaviour of the system to events that are safe. The work in [49], aimed generally at safe robotics, captures assumptions of the real world using Signal Temporal Logic (STL) and uses this for runtime monitoring of the robot.

Temporal logics have also been used to help overcome some of the challenges involved with developing swarm robotic systems. For example, Winfield et al. describe the notion of a *dependable swarm* as a distributed multi-robot system based upon the principles of swarm intelligence that we can place a high degree of reliance in [161]. They advocate the use of temporal logic as a means for specifying both safety and liveness properties of a simple wireless connected swarm. As part of their approach they discretise the robot's environment to a grid in order to simplify the requirements. Discretising the system specification is a common approach taken when using formal methods, since modelling continuous elements of the system is a difficult task.

As previously mentioned in §4.2, swarm robotic systems exhibit both macroscopic (at the level of the whole system) and microscopic (at the level of each robot) properties. The authors of [111] propose a novel specification language to allow the explicit specification both of the behaviour of the whole swarm and of individual robots. Their language captures a *region graph* of the environment of the swarm, *region propositions* that describe how the swarm can move through the environment, and LTL specifications of the macroscopic and microscopic behaviours of the swarm separately. These specifications are combined to produced decentralised controllers for the swarm robots.

The work in [30] aims to provide a development approach that avoids the trial and error approach of developing robot swarm behaviour from the bottom up. The approach begins with a PCTL specification of the swarm's macroscopic requirements. Successively more concrete models of the swarm are developed, including simulation and field testing implementations. Each step is checked against the previous steps to ensure the the swarm's behaviour matches the initial specification. However, this approach only focusses on the macroscopic level of the behaviour of the swarm, other approaches consider the microscopic level as well. The work in [92] presents a hierarchical framework that captures the (physical) dimensions of the swarm and uses LTL-X formulae to specify the behaviour. These formulae are used to project control laws for the individual robots in the swarm.

Bauer and Falcone proposed decentralised LTL monitors as a more efficient approach to monitoring in a distributed system (such as a robot swarm) [19]. This approach automatically distributes local monitors for particular components that determine the satisfaction or violation of a particular specification.

Swarm behaviours often lend themselves to probabilistic models, like in [101]. The work in [95] captures the probabilistic state machine from [101] using the input language for PRISM. The model is then checked for properties specified in PCTL. Since the swarm's behaviour can be characterised as a single state machine, replicated over each robot in the swarm, the authors use the *counting abstraction* to reduce the state space of their model. Instead of replicating the state machine for each robot in the swarm, each state maintains a counter of the number of robots in that sate. While a useful abstraction, this approach obviously only works when the swarm's behaviour is homogeneous.

Kouvaros and Lomuscio [98] use a fragment of the temporal-epistemic logic CTLK, without the 'next' operator (IACTLK-X) to model the alpha swarm-aggregation algorithm with an unknown number of agents. They then use MCMAS and a fault-injection approach to determine a lower bound on the number of faulty agents in a swarm, where the swarm can still satisfy its temporal-epistemic specification.

5.2 Discrete Event Systems

Petri Nets, FSMs – or Finite-State Automata (FSA) – are approaches to specifying behaviour based on discrete-state events. Formalisms for discrete-event systems include those capturing time (for example, TA or Timed Petri Nets) and probabilistic transitions (for example, PFSM or Generalised Stochastic Petri Net (GSPN)). These formalisms can be used to specify behaviour during the design phase, which can be checked for properties like deadlock freedom; or used as an input to a tool, which usually checks them for properties described in another formal language such as a temporal logic. There is a very close link between varieties of temporal logic and varieties of finite-state automata [148], and so many works combine both.

Petri Nets have had some use in modelling robotic systems. The work in [33] uses them to capture the abstract architecture of multiple reactive agents. These Petri Nets are then analysed for deadlock freedom of the nets, and therefore the underlying multi-agent system. In [162], Ziparo et al. extend Petri Nets to capture robot plans. The extension to Petri Nets is a set, G, of Goal Markings, which is proper subset of the reachable markings for the Petri Net that represents the desired result of the plan. The Petri Net Plans can be executed to find a sequence of transitions from the initial to the goal markings.

Costelha and Lima [41] use four layers of Petri Nets to model a robot's task plans, each layer describes a different element at a different level of abstraction. They use Marked Ordinary Petri Net (MOPN) to describe the robot's environment, enabling a more realistic model. Then, in order of increasing abstraction, they capture the actions a robot can take, the task plans, and the roles of various robots working together, using GSPNs. These models can be analysed for deadlocks and resource usage.

The model checker Uppaal uses networks of TA the describe input models, which are checked against properties written in a temporal logic. The work in [70] targets the ROS middleware framework, capturing the communication between ROS nodes using TA. The low-level information (for example, queue sizes and time outs) allows the verification of safety and liveness properties using Uppaal. Iftikhar and Weyns [81] use TA to model a decentralised traffic monitoring system, and TCTL to describe system invariants and correctness properties of the self-adaptive behaviour of this system. These properties are model checked using Uppaal.

The work in [103] uses FSA and a Gaussian model of the environment, based on a probability distribution of obstacle positions. These models are combined in MissionLab, a graphical robot mission designer. The robot FSA models are automatically translated into MissionLab's internal process algebra, Process Algebra for Robot Schemas (PARS).

The Automatic Modular Design approach (AutoMoDe) [62] is aimed at the development of swarm robotic systems. AutoMoDe generates a PFSM for each robot in a swarm. It optimises the PFSMs to produce individual robot behaviour that results in a swarm to achieve the required task.

In [78], Hoffmann et al. extend PFSMs with a weight function to map weights onto actions – which they call an APFSM. Because of the addition of the weight function, an APFSM is a discrete-time Markov chain. They use this formalism to describe autonomous behaviour and then use PRISM to verify properties about a pilotless aircraft on a foraging mission, with probabilistic modelling used to capture the physical environment.

5.3 Process Algebra

Process algebraic approaches define the behaviours of a system in terms of events and the interactions of processes. They are well suited to specifying concurrent systems. There are process algebras that can capture (discrete) time, but it seems that they do not often capture probabilistic behaviour.

Multi-agent systems can be described by a combination of the process algebra Finite State Processes (FSP) and π calculus combined with ADL (πADL) [6]. FSP is used to specify the system's required safety and liveness properties, which are transformed into Labelled-Transition Systems (LTSs); then the agent program and architecture (described in πADL) are checked to see if they satisfy the these properties.

Massink et al. [106] use the process algebra Bio-PEPA to model the microscopic behaviour of a robot swarm. Their choice of Bio-PEPA enables several different analysis methods using the same specification. They describe a specification of a robot swarm performing a foraging task where each object being collected requires a team of three robots, and the teams vote on taking either a long or short path between the start and pick-up points. They present results about the model using stochastic simulation, statistical model checking, and ordinary differential equations. Most interestingly, this collection of results is compared with existing analysis of the same case study; the comparison shows that Massink et al.'s approach provides similar results, but enables a wider range of analysis methods from a single specification.

RoboChart provides a formal semantics, based on CSP [76], for a timed state machine notation [128]. Each state in a state machine is modelled as the process $\mathit{Entry} \; ; \; (\mathit{During} \bigtriangleup \mathit{Transitions})$; where the *Entry* process captures the behaviour performed on entry to that state, followed sequentially (;) by the behaviour while in that state (*During*), which can be interrupted by ($\bigtriangleup$) a process capturing the outgoing transitions from that state (*Transitions*). This is a similar approach to the (non-formal) state chart notations ArmarX [150] and rFSM [107], described in §2. RoboChart is supported by an Eclipse-based environment, RoboTool [110], which allows the graphical construction of RoboChart diagrams that it can automatically translate into CSP, in the form described above. In CSP, events are instantaneous. RoboChart allows the specification of explicit time budgets and deadlines, which RoboTool translates into Timed CSP [127].

The CSP model-checker the Failures-Divergences Refinement checker (FDR) [66] can be opened from within RoboTool to enable quick checking of the timed and untimed properties of the model of the RoboChart diagram. RoboTool automatically generates basic assertions (that check for deadlock, livelock, timelock, and non-determinism) for FDR. These can be added to or edited, but this requires some skill with CSP and knowledge of the model, which many software engineers would not have. The ability to graphically edit and visualise the state machines is a key aspect of being able to integrate this notation into an existing engineering process.

The timed process algebra, CSP+T (which extends CSP with a notion of time), has also been used to capture the real-time aspects of robotic systems. The work in [5] derives correct and complete CSP+T specifications from a description of a system in the Unified Modelling Language (UML) profile, UML-RT – which is for developing real-time systems. This work captures each subsystem in the UML-RT description of the system, as a CSP+T process, and composes them in parallel. They demonstrate this approach on a two-armed industrial robot. However, they don't tackle the issue of how they ensure the correctness of their CSP+T specifications, with respect to the UML-RT models.

As previously mentioned in §5.2, the graphical robot mission design tool MissionLab contains a process algebra, PARS. The work in [103] describes a robot's behaviour using FSA and its environment using a Gaussian model, but MissionLab automatically translates these into PARS for checking the robot's behaviour within the given environment.

5.4 Model-Orientated Formalisms

Model-Orientated formalisms (for example B and Z) specify a system as a collection of data and a set of operations that change that data. They are well suited to capturing complicated data structures, but usually only provide limited features for capturing behaviour.

As previously mentioned in §4.3, self-adaptive (or reconfigurable) behaviour is often key for autonomous robotic systems. Weyns et al. describe a formal reference model for self-adaptive systems, called FORMS [157], which provides a Z model that can describe an arbitrary self-adaptive system. Weyns et al. suggest that this model could be used in conjunction with various existing tools to, for example: type check the adaptations to ensure that they are valid, visualise the model using animators, automatically generate test cases, or to transform the model into other notations to focus on other types of analysis. The model also provides a method of describing a self-adaptive system to enable communication about the adapting architecture during system design and comparison of different architectural configurations.

In [145], Event-B specifications are integrated with probabilistic properties to derive reconfigurable architectures for an on-board satellite system. The combination of these formalisms allows the models of reconfigurations to be checked – in PRISM – for both the derivation, via refinement, of the system from its specification, and the the probabilistic assessment of their reliability and performance.

Liang et al. [100] make use of the Jaza animator for Z specifications, combined with a Java debugger, to provide a runtime monitoring system. A Z specification describes the system's intended behaviour, the animation of this specification is compared to information from the ***jdb*** Java debugger to check that the running program implements the Z specification. They demonstrate this approach on a robotic assembly system, designed to build robots for the NASA ANTS project [3]. Two advantages of combining the Z animator with a Java debugger are that it keeps the monitor and program code separate, and that the approach doesn't require any alterations or additions to the monitored program.

5.5 Ontologies

An ontology can be used to formally specify the key concepts, properties, relationships and axioms of a given domain in such a way that it is possible to reason over the knowledge that it represents and infer new information. With respect to autonomous robotics, ontologies have been used to describe the robot environment, describe and/or reason about tasks/actions and for the reuse of domain knowledge [121]. In this sense, ontologies act as a body of knowledge and provide a vocabulary for a particular domain [69].

[3] `https://attic.gsfc.nasa.gov/ants/`

[10] uses ontological knowledge, represented using the Web Ontology Language (OWL), of a manufacturing environment to enable reconfiguration without any human intervention. This work makes use of OWL reasoners in order to decide how best to reconfigure the system.

KnowRob is a knowledge processing system for autonomous personal robotic assistants [146]. Here, knowledge is represented in description logic using OWL. Their system integrates encyclopaedic knowledge, an environment model, action-based reasoning and human observations. This approach can be used with ROS.

The IEEE-RAS working group been developing an ontology that is an explicit and formal specification of a shared conceptualisation for robotics and automation that has the form of a logical theory [121, 69]. This kind of formalism has the added benefit of being able to formally capture elements of Human-Robot Interaction.

5.6 Other Formalisms

Besides those described in the previous subsections, a number of other formalisms and approaches occur in the literature. We discuss these formal approaches below.

Similarly to the approaches (using temporal logics) described in [49] and [104], the work in [102] models the 'free' potential behaviour of the system separately to its intended behaviour. They adapt Supervisory Control Theory (SCT) to suit application to swarm robotics. SCT uses FSM-like models to capture the system's potential behaviour and then provides control specifications to constrain the system to its indented behaviours. This is related to the approach of runtime monitoring, discussed in §6.3.

In [155] and [152], Webster et al. use the BDI language GWENDOLEN [46] to build an agent to control an autonomous pilotless aircraft. They then verify LTL + Belief properties against the agent program, using AJPF. The LTL properties are based on the Rules of the Air and notions of Airmanship, which the Civil Aviation Authority (the UK's certification body for civilian aircraft) assume a pilot would follow. As previously mentioned in § 4.1, Webster et al. [152]'s pilotless agent program is also plugged into a flight simulator to provide a visualisation of the verified agent. Dennis et al. [45] present an approach that enables formal verification of the ethical choices of an autonomous system that is again controlled by a BDI agent, which specifically caters for the possibility of there being no ethical choice available to the system. This work extends GWENDOLEN with a logical framework for capturing ethical principles. Programs in this new agent language, ETHAN, can then be verified using AJPF [47]. Dennis et al. have used AJPF to verify that when an ETHAN agent chooses a plan, all other applicable plans are ethically worse than the chosen plan.

Gjondrekaj et al. [67] use a formal language that has been designed to capture properties about distributed systems, Klaim. They also use its stochastic extension, StoKlaim, and their related tool set. Their choice of language is influenced by their application domain: distributed robotic systems. They describe the modelling of three homogeneous robots, under distributed control, cooperating to transport objects.

An open problem in the sphere of agent-based systems is how to relate agent programs and agent logics. In this vein, Hindriks and Meyer have proposed an agent programming theory that consists of an agent programming language and a corresponding verification logic [75]. They utilise propositional dynamic logic for reasoning about actions and epistemic logic for reasoning about the knowledge of an agent. Then they formalise an operational semantics to relate the agent programming language to the agent logic. Here, the agent logic provides a declarative formalism to specify and reason about agents. This work provides a framework for employing and reasoning about programs written in an agent programming language.

In [108, 109] Mitsch et al. use dL [119], which was designed for the specification and verification of hybrid programs, to describe the discrete and continuous navigation behaviour of a ground robot. Because dL was designed for hybrid programs, it can describe the behaviour of a robotic system and the behaviour of a unknown environment. Mitsch et al. use the hybrid theorem prover KeYmaera [108] (and its sucessor, Keymaera X [109]) to prove safety (and liveness) properties about a robot's behaviour, written in dL. Mitsch et al. [109] also describe an automatic synthesis, from the verified dL model, of runtime monitors for safety properties.

5.7 Summary of Formalism

This section describes the formalisms currently used in the literature for specifying the behaviour of robotic systems. We have summarised the formalisms found using the methodology described in § 1.1 and their frequency in Table 2. These include temporal logics, discrete event system, process algebras, model-oriented specifications, ontologies and others. It is clear from the table that the use of temporal logics and discrete event systems has dominated the literature over the time period 2007–2018. Their prevalence could be explained by the popularity and usability of model-checking approaches that are amenable to temporal logics and discrete event systems which we will further discuss in the next section.

Approach	References	Total
Model Checking	[58], [153], [95], [154], [122], [42], [34], [112], [25], [81], [82], [21], [36], [106], [89]	17
Theorem Proving	[108]	1
Runtime Monitoring	[80], [54]	2
Integrated Formal Methods	[83],[36], [5]	2
Formal Software Frameworks/Architectures	[54], [125], [21], [158]	4
Reachability Analysis	[73]	1

Table 4: Summary of the approaches to formal verification found in the literature search. Note: some techniques use more than one approach.

6 Formal Verification Approaches for Robotics

This section discusses several different approaches to formal verification of robotic systems. Where § 6 discussed the *types* of formalisms found in the literature, this section discusses *how* these formalisms are used to verify robotic systems. If we imagine that the formalisms described in § 6 are different kinds of bricks, then the approaches we describe in this section are the type of building built using those bricks.

Table 4 shows the approaches that we found in the literature and the studies that use each approach; we discuss these approaches in this section. Section 6.1 describes model checking approaches, § 6.2 describes approaches using theorem proving, § 6.3 discusses approaches that use runtime verification and run-time monitors, § 6.4 discusses approaches that use a combination of different formalisms or approaches; finally, § 6.6 summarises the approaches found in the literature.

6.1 Model Checking

Model checking is the most widely used approach in verifying robotic systems; it has been used with temporal logics [51, 84], process algebras [110], and programs [58, 56]. Arguably, the popularity of model checking here owes something to the large number of publications that we found that use temporal logic, which are often used in model checking approaches. However, we see two main reasons why model checking may be *actively* chosen. First, model checkers are automatic tools, which makes them relatively easy to use; second, the concept of checking every state in a model to see if a required property holds is relatively easy to explain to stakeholders without formal methods experience, which may add to the wider confidence in results.

Some model checkers can handle timed models (such as Uppaal) or probabilistic models (such as PRISM) of a system's behaviour. This can be very useful for dealing with robotic systems; timing constraints are often required for safe behaviour and probability can be helpful when encoding the physical environment of the robot. Both of these features can improve confidence in the results. The input to a program model checker (such as AJPF) is the program code itself, so checking it for properties involves *symbolically executing* the code and assessing each execution against the required property [149].

Because model checking exhaustively explores the state space, one must be careful about the input models and properties to be checked. State explosion can be crippling to verification efforts that use model checking. For example, Webster et al. [153] report requiring 10^4 MB of memory was required to verify each individual safety property of their model of a domestic assistant robot. The work in [95] reports similar state explosion problems in models of swarm robotic systems. Various standard approaches can be used to mitigate state space explosion and keep models tractable [39, 38]. For example, for swarms with homogeneous behaviour approaches have been to use symmetry reduction [14] or abstract the swarm to a single state machine with a counting abstraction [152].

Model checking has also been used as an approach for building behaviour specifications, such as robot movement plans. For example, one approach uses model checking to build robot motion plans that satisfy properties specified in a deterministic subset of μ-calculus. A behavioural specification is built up incrementally, and checked after each expansion, until it moves the robot to its goal and satisfies the required properties. Kloetzer et al. [93] use model checking to find traces of a transition system describing the behaviour of a robot team that satisfy an LTL-X formula. The trace is then used to generate the communication and control strategy for each robot in the team.

Clearly model checking is a flexible approach: it has been used for a variety of formalisms that can describe concurrent, timed, and probabilistic systems; and has been used in verification efforts for a variety of robotic systems. The increasing access to computational power and memory, combined with clever ways to reduce the state space, maintain the popularity of model checking in the literature. The need to carefully craft a model for a particular model checker adds to the modelling time and reduces the portability of models between tools.

However, the automation and relative simplicity of the model checking approach to verifying robotic systems mean that it remains a focus for research.

6.2 Theorem Proving

Theorem proving approaches offer the benefit of producing a formal proof of the correctness of a software system. These formal proofs can be used to provide robust evidence for certification of autonomous robotic systems. A notable example here is the use of Isabelle/HOL and temporal logic to formalise a subset of traffic rules for vehicle overtaking in Germany [131].

Other work in this domain includes [108], which uses the hybrid systems logic dL [119] to describe the discrete and continuous navigation behaviour of a robot rover. They then use the hybrid theorem prover KeYmaera, Mitsch et al. to verify that the robot would not collide with stationary or moving obstacles, and maintains sufficient distance from obstacles. Mitsch et al. [109] update this work, verifying a less safety property that allows for imperfect sensors; adding liveness proofs, to guarantee progress; and automatically synthesising runtime monitors from the verified models, to mitigate the problems caused by the reality gap.

Although effective, it is clear from our analysis that these kinds of approaches have not received much attention in the literature. We believe that this is a usability issue with the tools generally more difficult to master than those of other approaches.

6.3 Runtime Monitoring

Runtime monitors can be used to sidestep the problem of verifying a (needfully) complex model of a robotic system. Instead of specifying and verifying the entire system, the properties that the system has to exhibit are extracted and specified as a monitor of the system. Because the monitor is simpler than the system, it is often easier to verify. Another advantage is that runtime monitors can mitigate the problem of the reality gap (between a model and the real world) especially when used to compliment offline verification. Given that a robotic system is naturally cyber-physical, and therefore malfunctions can have safety consequences, monitoring the system's behaviour at runtime can be key to safe operation [140].

A monitor consumes events from a system and compares them to the expected behaviour. If the system's events differ from the expectation, then it can invoke mitigating activities, including logging, flagging this to the user, or triggering behaviour to remedy the situation. Runtime monitoring cannot guarantee all system behaviours beforehand, it has received attention in the literature on on runtime verification, which brings together model-checking and stream-processing technologies [132]. Examples include [54, 81, 80], and the International Conference on Runtime Verification has been running since 2006.

Aniculaesei et al. [13] use runtime monitors to complement design-time formal methods. The runtime monitors are built to check that design-time assumptions hold during the execution of the program. For additional confidence at design-time, they also specify the system and its physical environment using TA with safety properties in TCTL.

Ferrando et al. [57] develop runtime verification to recognise anomalous environmental interactions and so highlight when the previous formal verification that has been carried out on an autonomous robotic system (which must have some environmental assumptions) becomes invalid.

Kane et al. [88] present runtime monitors for an autonomous research vehicle. The monitors are written in a safety specification language is called $\alpha\mathcal{VSL}$, the semantics of which is given over time-stamped traces. (In particular, it uses future-bounded, propositional Metric Temporal Logic (MTL).) They describe their algorithm EgMon which uses the MAUDE rewriting engine to reduce the input formulae. They extend EgMon with a hybrid monitoring algorithm, HMon, which first performs conservative checks and then performs as many eager checks as the remaining time permits.

Often, robotic systems are distributed; either because of the software architecture (such as Robot Operating System (ROS)) or because it is a multi-robot system (such as a robot swarm). For such systems, Bauer and Falcone [19] describe a method of distributing an LTL specification of a system's macro behaviour to a collection of micro-level behavioural monitors. Their approach does not assume any central information collection and ensure that the communication between monitors is minimal, but sufficient to allow the monitors to work.

Robotic systems are intrinsically cyber-physical, combining discrete and continuous parts. The implications of this on runtime monitoring are addressed in [140], which describes an approach for models the hybrid nature of cyber-physical systems without discretising them. They use Probabilistic Hybrid Automata (PHA) to capture both the discrete and continuous elements of a system. They provide an example of their approach using an electronically controlled train braking system. A similar hybrid-logic approach is taken by [109], who specify the safety properties of a robot rover using dL and then automatically synthesise runtime monitors from these verified models.

The work in [80] presents ROSRV, a runtime verification framework for robotics systems deployed on ROS. They describe as novel formal specification language that enables safety properties to be described. Using a new

ROS configuration file, ROSRV then automatically generates C++ ROS nodes to monitor the system for the specified properties. Because their approach uses normal ROS nodes and a custom configuration file, it doesn't require any changes to be make to either ROS or the application. They make the interesting observation that their approach cannot currently be verified because that "would require a model of ROS itself" to be able to prove that the monitors and other generated code respects the model of ROS.

Similarly, Falcone et al. [54] describe RV-BIP, a tool that produces runtime monitors for robots the BIP framework. They provide a formal description of the BIP component-based framework and monitors. Each monitor consumes events from the BIP system and assesses the sequence of events, producing a verdict of either true, false, or *currently* true or false. RV-BIPtakes an XML description of an LTS, describing the monitor, and produces a BIP monitor for a given BIP system. Falcone et al. provide an example where they use this tool to produce a BIP system with execution order and data freshness monitors.

Liang et al. [100] describe a runtime monitoring approach that does not require any alterations or additions to the monitored system. They specify the system in Z and compare the traces from a specification animator with information from a Java debugger to check if the running program is correct, with respect to the Z specification. This has the added advantage of decoupling the monitor from the monitored program.

6.4 Integrated Formal Methods

Integrated Formal Methods (iFM) refers to the integration of multiple formal methods, or a formal method with a non-formal approach, that complement each other. While still difficult, iFM can capture several dimensions of a system at once (for example state and dynamic behaviour) for easy analysis. In our previous work, we argue that robotic systems provide the impetus for addressing the challenges of integration; furthermore, because robotic systems are inherently hybrid, they *need* iFM [55]. An early example of this need is a study concluding that no single formalism was suitable for applying to a robot swam, and a combination of 4 formalisms would be the best approach [74].

The challenges the we discuss in § 4 are often best tackled using iFM. However, our literature search found no general approaches for using iFM for robotics; in this section we discuss some notable bespoke examples. To enable both refinement-based development and probabilistic checking, [145] uses both Event-B and a probabilistic language to check reconfigurable architectures for an on-board satellite system. The work in [6] combines FSP and πADL to capture safety and liveness properties of multi-agent robotic systems The FSP specifications of the relevant safety and liveness properties are transformed into LTSs, then the agent programs and architecture (described in πADL) are checked to see if they satisfy the required properties.

The work in [141] combines MAZE (an extension of Object-Z for multi-agent systems) and Back's action refinement to enable top-down development of robot swarms. In [86], the agent controlling the car is verified using the program model checker AJPF, and the timing properties are verified using Uppaal. This is extended, in [87] with a spatial reasoning calculus. The work verifies the cooperation between the vehicles, and the abstract behaviour of the physical vehicle. Another platooning driverless car is modelled using an integrated formalism combining CSP and B (CSP‖B) [40].

These bespoke examples integrate two or more formal methods to obtain an approach with their combined benefits. There are also examples that integrate a formalism with a non-formal notation. For example, an approach for deriving formal specifications from the real-time UML profile UML-RT [5], which captures UML-RT subsystems as processes in the timed CSP derivative CSP+T. This also provides UML-RT with a semantics in CSP+T.

Similarly, RoboChart is a robotic system design notation that integrates CSP with a graphical timed state machine notation [128]. The integration displayed by RoboChart provides a generic formal notation for robotic systems, which is an approach that yields a reusable and stable formal method. This approach to iFM – a reusable, stable, and generic method – is key to making iFMs work, not just for robotic systems but more generally as well. Integrating diverse formalisms into a a holistic framework remains an open problem, waiting to be solved.

Chkouri et al. present a metamodel-based from annex behaviour specification written in Architecture Analysis and Design Language (AADL) into the Behaviour Interaction Priority (BIP) framework. This facilitates model-checking, using the tool Aldebaran, in BIP for deadlock detection as well as the use of BIP observers to monitor safety [36]. This relates to approaches that build frameworks for robotic software engineering that enable or improve their verifiability, which we discuss next.

6.5 Frameworks for Verifiable Robotic Software

LTLMoP allows users to input structured english specifications that describe high-level robot behaviour and automatically generates hybrid controllers that can be used within a simulator and/or with physical robots [125]. LTL formulae are used to specify an automaton that is then used to generate the controllers. In this way, the resultant automaton is correct-by-construction.

The BIP framework [17] is a toolset for component-based design, it's notation is based on FSMs and it is used for modelling real-time software. The basic (atomic) component used is a state transition system, labelled with C/C++ functions. These components are combined to form larger components and models can be verified using the DFinder tool [21]. In [1], Abdellatif et al. combine the BIP framework with the G^{en}oM robot software architecture to provide a technique that can synthesise robotic control software from BIP models of the required behaviour. This enables the generation of robotic software, in G^{en}oM, that is correct by construction. Abdellatif et al. show the generation of software for a wheeled rover robot, using fault injection to test that the constrains in the BIP specification are enforced by the generated software.

Other work along this vein includes a metamodel-based translation from the AADL into BIP [36]. This facilitates simulation of AADL combined with formal verification using the model-checking tools in the BIP framework, in particular, this work uses the Aldebaran model-checker. Furthermore, Falcone et al. [54] integrate runtime verification with into the BIP framework. They describe their approach and a tool, RV-BIP, which generates monitors for BIP to check specifications at runtime.

The Averest framework provides tools for verifying temporal properties of synchronous programs that are written in the Quartz language [123]. These Quartz programs are translated into Averest Interchange Format (AIF) and then verified against LTL specifications using the Beryl model-checker or other third party tools that have interfaces to them such as SMV. Furthermore, the framework has a code generation tool called Topaz that can output Verilog, VHDL and C code.

Chen et al. have proposed a computational framework for automatic synthesis of control and communication strategies for robot teams using task specifications that are written as regular expressions [34]. Their objective is to generate provably correct individual control and communication strategies from regular expressions that are used to create finite state automaton (FSA). Their approach is illustrated using a robotic urban-like environment where autonomous cars must navigate and avoid collisions.

The FOrmal Reference Model for Self-adaptation (FORMS) provides a reference model that can guide developers as to how to formulate a Z specification of a self-adaptive system [158]. This is built upon MAPE-K and supports agents and formal refinement of specifications in Z.

Kim et al. [91] employ the Esterel framework to verify the stopping behaviour of a home service robot. This framework offers facilities for specifying and verifying robotic systems using model checking. Esterel can also be translated into an associated C or C++ program and the semantics of an Esterel program is given in terms of the FSM that it describes. To enable systems to be checked for temporal logic properties, the authors translate them into observers in Esterel.

Others that should be included are R^{2}C which is a state checker used in the LAAS architecture, the ORCAAD system which is based on the Esterel language.

6.6 Summary of Approaches

In this section we have summarised the approaches taken throughout our literature survey in Table 4. This table indicates that there is a current prevalence of model-checking techniques for the formal specification and verification of autonomous robotic systems. This was also reflected by the dominance of temporal logics and discrete event systems in Table 2. It is also interesting to note that there have been a number of attempts to devise a formal framework or architecture for developing these systems that may often include integration with model-checkers.

6.6.1 Answering RQ2 and RQ3:

Our second research question, RQ2, asked what the current best practice formalisms, tools and approaches are for specifying and verifying the behaviour of autonomous robotic systems. It is clear from our analysis of the literature over the last decade that temporal logics, discrete event systems and model-checkers currently dominate this field. This is most likely due to their ease of use.

RQ3 asks what are the limitations of these formalisms, tools and approaches. Model-checking is a powerful tool for the verification of autonomous systems. However, due to state space explosion and the error prone process of manual translation of software into models, it has not had much impact on the way that autonomous systems are designed and developed [139]. The idea of a formal framework for developing robotic systems in a way that facilitates formal specification and verification is attractive but it involves interfacing to a number of different verification tools that do not speak the same language. In this scenario some kind of a translation step is required and the translations themselves tend to lack a formal underpinning.

7 Discussion

This paper presented a survey of the literature pertaining to the formal specification and verification of autonomous robotic systems. It began by outlining our methodology and identifying three research questions, RQ1, RQ2 and RQ3, in §1.1. We have explored these questions in detail and analysed the results throughout this document.

RQ1 sought to identify the major challenges to the formal specification and verification of autonomous robotic systems. We partitioned these into two sets; those that are external to the robotic system and those that are internal, or part of, the robotic system. These external challenges amounted to the modelling of the robot's physical environment and the provision of sufficient evidence for certification and obtaining trust. The internal challenges consist of those robotic systems that are agent based, homogeneous multi-robot, heterogeneous multi-robot and those that are self-adaptive and reconfigurable.

RQ2 concerned the current state-of-the-art formalisms, tools and approaches to formal specification and verification of autonomous robotic systems. To this end, our analysis revealed that temporal logics, discrete event systems and model-checking approaches were paramount here. As a follow on from RQ2, RQ3 analysed the current limitations of these approaches which centres mainly on the failure of model-checking techniques as the state space of these complex systems grows. Further to this formal frameworks that facilitate and support formal specification and verification are emerging, although still in their infancy. Following this strain of research, it is clear that no one formalism/tool/logic can ensure the correctness of an autonomous robotic system in its entirety so more work must be done on integrating the formal methods in practice.

References

[1] Tesnim Abdellatif, Saddek Bensalem, Jacques Combaz, Lavindra De Silva, and Felix Ingrand. Rigorous design of robot software: A formal component-based approach. *Rob. Auton. Syst.*, 60(12):1563–1578, 2012.

[2] Kai Adam, Katrin Olldobler, Bernhard Rumpe, and Andreas Wortmann. Modeling Robotics Software Architectures with Modular Model Transformations. *J. Softw. Eng. Robot.*, 8(1):3–16, apr 2017.

[3] Sorin Adam, Morten Larsen, Kjeld Jensen, and Ulrik Pagh Schultz. Rule-based Dynamic Safety Monitoring for Mobile Robots. *Journal of Software Engineering for Robotics*, 7(1):120–141, 2016.

[4] Ho Seok Ahn, Min Ho Lee, and Bruce A MacDonald. Design of a Robotic Software Manager for using Heterogeneous Components of Different Frameworks. *Journal of Software Engineering for Robotics*, 8(1):45–64, 2017.

[5] K Benghazi Akhlaki, M I Capel Tuñón, J A Holgado Terriza, L E Mendoza Morales, K. Benghazi Akhlaki, M. I. Capel Tuñón, J A Holgado Terriza, L. E. Mendoza Morales, K Benghazi Akhlaki, M I Capel Tuñón, J A Holgado Terriza, and L E Mendoza Morales. A methodological approach to the formal specification of real-time systems by transformation of UML-RT design models. *Science of Computer Programming*, 65(1):41–56, 2007.

[6] Nadeem Akhtar. Contribution to the Formal Specification and Verification of a Multi-Agent Robotic System. *European Journal of Scientific Research*, 117(1):2014, 2014.

[7] Rachid Alami, Raja Chatila, Sara Fleury, Malik Ghallab, and Flix Ingrand. An architecture for autonomy. *The International Journal of Robotics Research*, 17(4):315–337, 1998.

[8] Rachid Alami, K. Madhava Krishna, and Thierry Siméon. Provably Safe Motions Strategies for Mobile Robots in Dynamic Domains. In *Auton. Navig. Dyn. Environ.*, volume 35, pages 85–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[9] Sonya Alexandrova, Zachary Tatlock, and Maya Cakmak. RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In *Int. Conf. Robot. Autom.*, pages 5537–5544. IEEE, may 2015.

[10] Yazen Alsafi and Valeriy Vyatkin. Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing. *Robotics and Computer-Integrated Manufacturing*, 26(4):381–391, 2010.

[11] Matthias Althoff, Daniel Althoff, Dirk Wollherr, and Martin Buss. Safety verification of autonomous vehicles for coordinated evasive maneuvers. In *Intell. Veh. Symp.*, pages 1078–1083. IEEE, jun 2010.

[12] Noriaki Ando, Takashi Suehiro, and Tetsuo Kotoku. A software platform for component based RT-system development: OpenRTM-Aist. In Stefano Carpin, Itsuki Noda, Enrico Pagello, Monica Reggiani, and Oskar von Stryk, editors, *LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5325 LNAI, pages 87–98, Berlin, Heidelberg, 2008. Springer.

[13] Adina Aniculaesei, Daniel Arnsberger, Falk Howar, and Andreas Rausch. Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments. *Electron. Proc. Theor. Comput. Sci.*, 232:79–90, dec 2016.

[14] Laura Antuña, Dejanira Araiza-Illan, Sérgio Campos, and Kerstin Eder. Symmetry reduction enables model checking of more complex emergent behaviours of swarm navigation algorithms. In *Lect. Notes Comput. Sci.*, volume 9287 of *LNCS*, pages 26–37. Springer, Cham, 2015.

[15] Ronald C. Arkin, Patrick Ulam, and Brittany Duncan. An Ethical Governor for Constraining Lethal Action in an Autonomous System. Technical Report Technical Report GIT-GVU-09-02, GEORGIA INST OF TECH ATLANTA MOBILE ROBOT LAB, GEORGIA, 2009.

[16] Ronald Craig Arkin, Patrick Ulam, and Alan R. Wagner. Moral Decision Making in Autonomous Systems: Enforcement, Moral Emotions, Dignity, Trust, and Deception. *Proc. IEEE*, 100(3):571–589, mar 2012.

[17] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Int. Conf. Softw. Eng. Form. Methods*, pages 3–12, Pune, India, 2006. IEEE.

[18] Ananda Basu, Matthieu Gallien, Charles Lesire, Thanh-Hung Nguyen, Saddek Bensalem, Félix Ingrand, and Joseph Sifakis. Incremental component-based construction and verification of a robotic system. In *ECAI*, volume 178, pages 631–635, 2008.

[19] Andreas Bauer and Yliès Falcone. Decentralised LTL Monitoring. *Form. Methods Syst. Des.*, 48(1-2):46–93, apr 2016.

[20] Gerardo Beni. From Swarm Intelligence to Swarm Robotics. In Erol Şahin and William M Spears, editors, *Swarm Robot.*, pages 1–9, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[21] Saddek Bensalem, Lavindra De Silva, Andreas Griesmayer, Felix Ingrand, Axel Legay, and Rongjie Yan. A formal approach for incremental construction with an application to autonomous robotic systems. In *International Conference on Software Composition*, pages 116–132. Springer, 2011.

[22] ZM Bi, Sherman YT Lang, M Verner, and P Orban. Development of reconfigurable machines. *The International Journal of Advanced Manufacturing Technology*, 39(11-12):1227–1251, 2008.

[23] Geoffrey Biggs and Bruce MacDonald. Specifying Robot Reactivity in Procedural Languages. In *2006 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pages 3735–3740. IEEE, oct 2006.

[24] H. A. P. Blom and J. Lygeros, editors. *Stochastic Hybrid Systems*, volume 337 of *Lecture Notes in Control and Information Science*. Springer, 2006.

[25] Rafael H Bordini, Michael Fisher, and Maarten Sierhuis. Formal verification of human-robot teamwork. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pages 267–268. ACM, 2009.

[26] Sara Bouraine, Thierry Fraichard, and Hassen Salhi. Provably safe navigation for mobile robots with limited field-of-views in dynamic environments. *Autonomous Robots*, 32(3):267–283, 2012.

[27] M. Bozzano, A. Cimatti, M. Roveri, and A. Tchaltsev. A comprehensive approach to on-board autonomy verification and validation. In *IJCAI Int. Jt. Conf. Artif. Intell.*, volume 11, pages 2398–2403, Barcelona, Catalonia, Spain, 2011.

[28] Ronen I. Brafman, Michael Bar-Sinai, and Maor Ashkenazi. Performance level profiles: A formal language for describing the expected performance of functional modules. In *Int. Conf. Intell. Robot. Syst.*, volume 2016-Novem, pages 1751–1756. IEEE, oct 2016.

[29] Julia M. B. Braman, Richard M. Murray, and David A. Wagner. Safety verification of a fault tolerant reconfigurable autonomous goal-based robotic control system. In *Int. Conf. Intell. Robot. Syst.*, pages 853–858. IEEE, IEEE, oct 2007.

[30] Manuele Brambilla, Arne Brutschy, Marco Dorigo, and Mauro Birattari. Property-Driven Design for Robot Swarms. *ACM Trans. Auton. Adapt. Syst.*, 9(4):1–28, 2014.

[31] Donald P Brutzman, Duane T Davis, George R Lucas, and Robert B McGhee. Run-time Ethics Checking for Autonomous Unmanned Vehicles: Developing a Practical Approach. In *Int. Symp. Unmanned Untethered Submers. Technol.*, Portsmouth New Hampshire, 2013. Monterey, California: Naval Postgraduate School.

[32] Herman Bruyninckx, Markus Klotzbücher, Nico Hochgeschwender, Gerhard Kraetzschmar, Luca Gherardi, and Davide Brugali. The BRICS component model. In *Proc. 28th Annu. ACM Symp. Appl. Comput.*, page 1758, New York, New York, USA, 2013. ACM Press.

[33] Jose R Celaya, Alan a Desrochers, and Robert J Graves. Modeling and analysis of multi-agent systems using petri nets. In Eugene Santos, Liping Fang, Andreas Nürnberger, Mak Tafazoli, and Hideyuki Takagi, editors, *2007 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 1439–1444, Montreal, Canada, oct 2007. IEEE, IEEE.

[34] Yushan Chen, Xu Chu Ding, Alin Stefanescu, and Calin Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2012.

[35] Betty H. C. Cheng, Kerstin I. Eder, Martin Gogolla, Lars Grunske, Marin Litoiu, Hausi A. Müller, Patrizio Pelliccione, Anna Perini, Nauman A. Qureshi, Bernhard Rumpe, Daniel Schneider, Frank Trollmann, and Norha M. Villegas. Using Models at Runtime to Address Assurance for Self-Adaptive Systems. In *Lect. Notes Comput. Sci.*, volume 8378 LNCS of *LNCS*, pages 101–136. Springer, Cham, 2014.

[36] M Yassin Chkouri, Anne Robert, Marius Bozga, and Joseph Sifakis. Translating aadl into bip-application to the verification of real-time systems. In *International Conference on Model Driven Engineering Languages and Systems*, pages 5–19. Springer, 2008.

[37] Jiyoung Choi, Seungkeun Kim, and Antonios Tsourdos. Verification of heterogeneous multi-agent system using mcmas. *International Journal of Systems Science*, 46(4):634–651, 2015.

[38] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *JACM*, 50(5):752–794, 2003.

[39] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model Checking and Abstraction. *ACM Trans. Programming Languages and Systems*, 16(5):1512–1542, 1994.

[40] Samuel Colin, Arnaud Lanoix, Olga Kouchnarenko, and Jeanine Souquières. Using CSP——B Components: Application to a Platoon of Vehicles. In *Form. Methods Ind. Crit. Syst.*, volume 5596 of *LNCS*, pages 103–118, 2009.

[41] Hugo Costelha and Pedro Lima. Modelling, analysis and execution of multi-robot tasks using petri nets. In Thomas C. Henderson and Edward Grant, editors, *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, pages 1187–1190, San Diego, USA, oct 2008. IEEE, IEEE.

[42] Neil Dantam and Mike Stilman. Robust and efficient communication for real-time multi-process robot software. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 316–322. IEEE, 2012.

[43] Ewen Denney and Ganesh Pai. Automating the assembly of aviation safety cases. *IEEE Transactions on Reliability*, 63(4):830–849, 2014.

[44] Ewen Denney and Steven Trac. A software safety certification tool for automatically generated guidance, navigation and control code. In *Aerospace Conference*, pages 1–11. IEEE, 2008.

[45] Louise Dennis, Michael Fisher, Marija Slavkovik, and Matt Webster. Formal verification of ethical choices in autonomous systems. *Robotics and Autonomous Systems*, 2016.

[46] Louise a Dennis and Berndt Farwer. Gwendolen : A BDI Language for Verifiable Agents. *Logic and the simulation of interaction and reasoning*, 2008.

[47] Louise A. Dennis, Michael Fisher, Matthew P. Webster, and Rafael H. Bordini. Model checking agent programming languages. *Automated Software Engineering*, 19(1):5–63, 2012.

[48] Louise A Dennis, Michael Fisher, and Alan F T Winfield. Towards Verifiably Ethical Robot Behaviour. *CoRR*, abs/1504.0(2014):1–11, 2015.

[49] Ankush Desai, Tommaso Dreossi, and Sanjit A. Seshia. Combining Model Checking and Runtime Verification for Safe Robotics. In *RV 2017*, volume 10548, pages 172–189, 2017.

[50] Mark d'Inverno, Michael Luck, Michael Georgeff, David Kinny, and Michael Wooldridge. The dmars architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):5–53, 2004.

[51] Clare Dixon, Matt Webster, Joe Saunders, Michael Fisher, and Kerstin Dautenhahn. "The fridge door is open" - Temporal verification of a robotic assistant's behaviours. *LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8717 LNAI:97–108, 2014.

[52] Marco Dorigo and Erol Şahin. Guest Editorial. *Autonomous Robots*, 17(2-3):111–113, sep 2004.

[53] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus. Architecture-driven self-adaptation and self-management in robotics systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 142–151, May 2009.

[54] Yliès Falcone, Mohamad Jaber, Thanh-hung Nguyen, Marius Bozga, and Saddek Bensalem. Runtime Verification of Component-Based Systems. In G Barthe, A Pardo, and G Schneider, editors, *Softw. Eng. Form. Methods*, pages 204–220, Berlin, Heidelberg, 2011. Springer, Springer, Berlin, Heidelberg.

[55] Marie Farrell, Matt Luckcuck, and Michael Fisher. Robotics and Integrated Formal Methods: Necessity meets Opportunity. In *Integr. Form. Methods*, page 10, [Accepted].

[56] Lucas E. R. Fernandes, Vinicius Custodio, Gleifer V. Alves, and Michael Fisher. A Rational Agent Controlling an Autonomous Vehicle: Implementation and Formal Verification. *Electronic Proceedings in Theoretical Computer Science*, 257(Fvav):35–42, 2017.

[57] Angelo Ferrando, Louise A. Dennis, Davide Ancona, Michael Fisher, and Viviana Mascardi. "recognising assumption violations in autonomous systems verificaion". In *Proceedings of the 2018 International Conference on Autonomous Agents and Multiagent Systems*, Stockholm, Sweden, 2018. To Appear.

[58] Michael Fisher, Louise A. Dennis, and Matt Webster. Verifying autonomous systems. *Communications of the ACM*, 56(9):84–93, 2013.

[59] Franck Fleurey, Benoit Baudry, Robert France, and Sudipto Ghosh. A generic approach for automatic model composition. In *International Conference on Model Driven Engineering Languages and Systems*, pages 7–15. Springer, 2007.

[60] Sara Fleury, Matthieu Herrb, and Raja Chatila. G$^{\text{en}}$oM: A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 842 – 849 vol.2. IEEE, 1997.

[61] Mohammed Foughali, Bernard Berthomieu, Silvano Dal Zilio, Félix Ingrand, and Anthony Mallet. Model Checking Real-Time Properties on the Functional Layer of Autonomous Robots. In Kazuhiro Ogata, Mark Lawford, and Shaoying Liu, editors, *Formal Methods and Software Engineering*, pages 383–399, Cham, 2016. Springer International Publishing.

[62] Gianpiero Francesca, Manuele Brambilla, Arne Brutschy, Vito Trianni, and Mauro Birattari. AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112, 2014.

[63] Paul Gainer, Clare Dixon, Kerstin Dautenhahn, Michael Fisher, Ullrich Hustadt, Joe Saunders, and Matt Webster. CRutoN: Automatic Verification of a Robotic Assistant's Behaviours. In Laure Petrucci, Cristina Seceleanu, and Ana Cavalcanti, editors, *Critical Systems: Formal Methods and Automated Verification*, pages 119–133, Cham, 2017. Springer International Publishing.

[64] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. The belief-desire-intention model of agency. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 1–10. Springer, 1998.

[65] Michael P Georgeff and Amy L Lansky. Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682, 1987.

[66] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A W Roscoe. FDR3 — A Modern Model Checker for CSP. In *Tools Algorithms Constr. Anal. Syst.*, volume 8413 of *LNCS*, pages 187–201, 2014.

[67] Edmond Gjondrekaj, Michele Loreti, Rosario Pugliese, Francesco Tiezzi, Carlo Pinciroli, Manuele Brambilla, Mauro Birattari, and Marco Dorigo. Towards a formal verification methodology for collective robotic systems. In Kenji Taguchi and Toshiaki Aoki, editors, *International Conference on Formal Engineering Methods*, volume 7635 LNCS, pages 54–70, Kyoto, Japan, 2012. Springer.

[68] Jérémie Guiochet, Mathilde Machin, and Hélène Waeselynck. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 94:43–52, 2017.

[69] Tamás Haidegger, Marcos Barreto, Paulo Gonçalves, Maki K Habib, Sampath Kumar Veera Ragavan, Howard Li, Alberto Vaccarella, Roberta Perrone, and Edson Prestes. Applied ontologies and standards for service robots. *Robotics and Autonomous Systems*, 61(11):1215–1223, 2013.

[70] Raju Halder, José Proença, Nuno MacEdo, and André Santos. Formal verification of ROS-based robotic applications using timed-automata. *Proceedings - 2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering, FormaliSE 2017*, pages 44–50, 2017.

[71] David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231 – 274, 1987.

[72] Benjamin Herd, Simon Miles, Peter Mcburney, and Michael Luck. An LTL-based property specification language for agent-based simulation traces. Technical Report Technical Report 14-02 Abstract., King's College London, London, 2014.

[73] Daniel Heß, Matthias Althoff, and Thomas Sattel. Formal verification of maneuver automata for parameterized motion primitives. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1474–1481. IEEE, 2014.

[74] Mike Hinchey, James L. Rash, Christopher A. Rouff, Walt F. Truszkowski, and Amy K. C. S. Vanderbilt. Requirements of an Integrated Formal Method for Intelligent Swarms. In *Form. Methods Ind. Crit. Syst.*, pages 33–59. John Wiley & Sons, Inc., Hoboken, NJ, USA, nov 2012.

[75] Koen V Hindriks and J-J Ch Meyer. Toward a programming theory for rational agents. *Autonomous Agents and Multi-Agent Systems*, 19(1):4–29, 2009.

[76] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[77] CAR Hoare. Viewpoint retrospective: An axiomatic basis for computer programming. *Communications of the ACM*, 52(10):30–32, 2009.

[78] Ruth Hoffmann, Murray Ireland, Alice Miller, Gethin Norman, and Sandor Veres. Autonomous agent behaviour modelled in PRISM A case study. In D Bošnački and A Wijs, editors, *International Symposium on Model Checking Software*, volume 9641, pages 104–110, Cham, 2016. Springer, Cham.

[79] Yingbing Hua, Stefan Zander, Mirko Bordignon, and Bjorn Hein. From AutomationML to ROS: A model-driven approach for software engineering of industrial robotics using ontological reasoning. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 2016-Novem, pages 1–8. IEEE, sep 2016.

[80] Jeff Huang, Cansu Erdogan, Yi Zhang, Brandon Moore, Qingzhou Luo, Aravind Sundaresan, and Grigore Rosu. Rosrv: Runtime verification for robots. In *International Conference on Runtime Verification*, pages 247–254. Springer, 2014.

[81] M. Usman Iftikhar and Danny Weyns. A Case Study on Formal Verification of Self-Adaptive Behaviors in a Decentralized System. *Electron. Proc. Theor. Comput. Sci.*, 91:45–62, aug 2012.

[82] M. Usman Iftikhar and Danny Weyns. ActivFORMS: active formal models for self-adaptation. In Gregor Engels and Nelly Bencomo, editors, *Proc. 9th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst. - SEAMS 2014*, pages 125–134, New York, New York, USA, 2014. ACM, ACM Press.

[83] Didac Gil De La Iglesia and Danny Weyns. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3):15, 2015.

[84] Paolo Izzo, Hongyang Qu, and Sandor M. Veres. A stochastically verifiable autonomous control architecture with reasoning. *2016 IEEE 55th Conference on Decision and Control, CDC 2016*, pages 4985–4991, 2016.

[85] Choulsoo Jang, Seung Ik Lee, Seung Woog Jung, Byoungyoul Song, Rockwon Kim, Sunghoon Kim, and Cheol Hoon Lee. Opros: A new component-based robot software platform. *ETRI Journal*, 32(5):646–656, 2010.

[86] Maryam Kamali, Louise A. Dennis, Owen McAree, Michael Fisher, and Sandor M. Veres. Formal Verification of Autonomous Vehicle Platooning. *Sci. Comput. Program.*, 148:88–106, 2017.

[87] Maryam Kamali, Sven Linker, and Michael Fisher. Modular Verification of Vehicle Platooning with Respect to Decisions, Space and Time. apr 2018.

[88] Aaron Kane, Omar Chowdhury, Anupam Datta, and Philip Koopman. A case study on runtime monitoring of an autonomous research vehicle (arv) system. In *Runtime Verification*, pages 102–117. Springer, 2015.

[89] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In John Baillieul and Lei Guo, editors, *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 2222–2229, Shanghai, China, dec 2009. IEEE, IEEE.

[90] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[91] Moonzoo Kim, Kyo Chul Kang, and Hyoungki Lee. Formal verification of robot movements-a case study on home service robot shr100. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4739–4744. IEEE, 2005.

[92] Marius Kloetzer and Calin Belta. Temporal logic planning and control of robotic swarms by hierarchical abstractions. *IEEE Transactions on Robotics*, 23(2):320–330, 2007.

[93] Marius Kloetzer, Xu Chu Ding, and Calin Belta. Multi-robot deployment from LTL specifications with reduced communication. In Marios M. Polycarpou, editor, *IEEE Conference on Decision and Control and European Control Conference*, pages 4867–4872, Orlando, USA, 2011. IEEE.

[94] John C Knight, Colleen L Dejong, Matthew S Gibble, and Luís G Nakano. Why Are Formal Methods Not Used More Widely ? In *NASA Langley Formal Methods Workshop*, page 12, Hampton, VA, USA, 1997. Virginia Univ.; Dept. of Computer Science; Charlottesville, VA United States.

[95] Savas Konur, Clare Dixon, and Michael Fisher. Formal Verification of Probabilistic Swarm Behaviours. In Marco Dorigo, Mauro Birattari, Gianni A Di Caro, René Doursat, Andries P Engelbrecht, Dario Floreano, Luca Maria Gambardella, Roderich Groß, Erol \cSahin, Hiroki Sayama, and Thomas Stützle, editors, *Swarm Intelligence*, volume 6234 LNCS of *LNCS*, pages 440–447. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[96] Felix Kossak and Atif Mashkoor. How to select the suitable formal method for an industrial application: A survey. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 213–228. Springer, 2016.

[97] Yanni Kouskoulas, David Renshaw, André Platzer, and Peter Kazanzides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 263–272. ACM, 2013.

[98] Panagiotis Kouvaros and Alessio Lomuscio. Verifying fault-tolerance in parameterised multi-agent systems. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 288–294, 2017.

[99] Hadas Kress-Gazit, Tichakorn Wongpiromsarn, and Ufuk Topcu. Correct, Reactive, High-Level Robot Control. *IEEE Robotics & Automation Magazine*, 18(3):65–74, sep 2011.

[100] Hui Liang, Jin Song Dong, Jing Sun, and W. Eric Wong. Software monitoring through formal specification animation. *Innovations in Systems and Software Engineering*, 5(4):231–241, dec 2009.

[101] Wenguo Liu, Alan F T Winfield, and Jin Sa. Modelling Swarm Robotic Systems: A Case Study in Collective Foraging. In *Proceedings of Towards Autonomous Robotic Systems*, pages 25–32, 2007.

[102] Yuri K Lopes, Stefan M Trenkwalder, · André, B Leal, Tony J Dodd, Roderich Groß, and André B Leal. Supervisory control theory applied to swarm robotics. *Swarm Intelligence*, 10:65–97, 2016.

[103] Damian M. Lyons, Ronald C. Arkin, Shu Jiang, Dagan Harrington, Feng Tang, and Peng Tang. Probabilistic verification of multi-robot missions in uncertain environments. In *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, volume 2016-Janua, pages 56–63, 2016.

[104] Mathilde Machin, Fanny Dufossé, Jean-paul Blanquart, Jérémie Guiochet, David Powell, and Hélène Waeselynck. Specifying Safety Monitors for Autonomous Systems Using Model-Checking. In Andrea Bondavalli and Felicita Di Giandomenico, editors, *Comput. Safety, Reliab. Secur.*, volume 8666 of *LNCS*, pages 262–277. Springer International Publishing, Cham, 2014.

[105] Mathilde Machin, Fanny Dufosse, Jeremie Guiochet, David Powell, Matthieu Roy, and Helene Waeselynck. Model-Checking and Game theory for Synthesis of Safety Rules. In Remzi Seker and Kenji Yoshigoe, editors, *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, number January in International Symposium on High Assurance Systems Engineering, pages 36–43, Daytona Beach Shores, USA, jan 2015. IEEE.

[106] Mieke Massink, Manuele Brambilla, Diego Latella, Marco Dorigo, and Mauro Birattari. On the use of Bio-PEPA for modelling and analysing collective behaviours in swarm robotics. *Swarm Intelligence*, 7(2-3):201–228, sep 2013.

[107] T. Merz, P. Rudol, and M. Wzorek. Control system framework for autonomous robots based on extended state machines. In *International Conference on Autonomic and Autonomous Systems (ICAS'06)*, pages 14–14, July 2006.

[108] Stefan Mitsch, Khalil Ghorbal, and Andre Platzer. On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles. In *Robotics: Science and Systems IX*, page 8. Robotics: Science and Systems Foundation, jun 2013.

[109] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *Int. J. Rob. Res.*, 36(12):1312–1340, 2017.

[110] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, and Jon Timmis. Automatic property checking of robotic applications. In *Int. Conf. Intell. Robot. Syst.*, volume 2017-Septe, pages 3869–3876. IEEE, sep 2017.

[111] Salar Moarref and Hadas Kress-Gazit. Decentralized control of robotic swarms from high-level temporal logic specifications. In *International Symposium on Multi-Robot and Multi-Agent Systems*, pages 17–23. IEEE, 2017.

[112] Levente Molnar and SM Veres. System verification of autonomous underwater vehicles by model checking. In *OCEANS 2009-EUROPE*, pages 1–10. IEEE, 2009.

[113] Jeremy Morse, Dejanira Araiza-Illan, Jonathan Lawry, Arthur Richards, and Kerstin Eder. Formal Specification and Analysis of Autonomous Systems under Partial Compliance. mar 2016.

[114] Arne Nordmann, Nico Hochgeschwender, Dennis Wigand, and Sebastian Wrede. A Survey on Domain-Specific Languages in Robotics. *Journal of Software Engineering for Robotics*, 7(July):75–99, 2016.

[115] Matthew O'Brien, Ronald C Arkin, Dagan Harrington, Damian Lyons, and Shu Jiang. Automatic Verification of Autonomous Robot Missions. In Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald, editors, *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, volume LNCS vol 8810 of *Lecture Notes in Computer Science*, pages 462–473. Springer International Publishing, Cham, 2014.

[116] P. Tabdada. *Verification and Control of Hybrid Systems: A Symbolic Approach.* Springer, 2009.

[117] Lucia Pallottino, Vincenzo G Scordio, Antonio Bicchi, and Emilio Frazzoli. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics*, 23(6):1170–1183, 2007.

[118] Dung Phan, Junxing Yang, Denise Ratasich, Radu Grosu, Scott A Smolka, and Scott D Stoller. Collision avoidance for mobile robots with limited sensing and limited information about the environment. In *Runtime Verification*, pages 201–215. Springer, 2015.

[119] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In Nicola Olivetti, editor, *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007.

[120] Rodion Podorozhny, Sarfraz Khurshid, Dewayne Perry, and Xiaoqin Zhang. Verification of multi-agent negotiations using the alloy analyzer. In *International Conference on Integrated Formal Methods*, pages 501–517. Springer, 2007.

[121] Edson Prestes, Joel Luis Carbonera, Sandro Rama Fiorini, Vitor A. M. Jorge, Mara Abel, Raj Madhavan, Angela Locoro, Paulo Goncalves, Marcos E. Barreto, Maki Habib, Abdelghani Chibani, Sbastien Grard, Yacine Amirat, and Craig Schlenoff. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193 – 1204, 2013. Ubiquitous Robotics.

[122] Martin Proetzsch, Karsten Berns, Tobias Schuele, and Klaus Schneider. Formal verification of safety behaviours of the outdoor robot ravon. In *ICINCO-RA (1)*, pages 157–164, 2007.

[123] Martin Proetzsch, Tobias Luksch, and Karsten Berns. The behaviour-based control architecture ib2c for complex robotic systems. In *Annual Conference on Artificial Intelligence*, pages 494–497. Springer, 2007.

[124] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In Kazuhiro Kosuge and Katsushi Ikeuchi, editors, *ICRA workshop on open source software*, volume 3, page 5, Kobe, Japan, 2009. IEEE ICRA.

[125] Vasumathi Raman and Hadas Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using ltlmop. In *International Conference on Computer Aided Verification*, pages 663–668. Springer, 2011.

[126] A. S. Rao and M. Georgeff. BDI Agents: From Theory to Practice. In *Proc. 1st Int. Conf. Multi-Agent Systems (ICMAS)*, pages 312–319, San Francisco, USA, June 1995.

[127] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 226 LNCS:314–323, 1986.

[128] Pedro Ribeiro, Alvaro Miyazawa, Wei Li, Ana Cavalcanti, and Jon Timmis. Modelling and verification of timed robotic controllers. *LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10510 LNCS:18–33, 2017.

[129] Jan Oliver Ringert, Alexander Roth, Bernhard Rumpe, and Andreas Wortmann. Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems. *J. Softw. Eng. Robot.*, 6(1):33–57, 2015.

[130] Max Risler and Oskar von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in xabsl. In *AAMAS08-workshop on formal models and methods for multi-robot systems, Estoril, Portugal.* Citeseer, 2008.

[131] Albert Rizaldi, Jonas Keinholz, Monika Huber, Jochen Feldle, Fabian Immler, Matthias Althoff, Eric Hilgendorf, and Tobias Nipkow. Formalising and monitoring traffic rules for autonomous vehicles in isabelle/hol. In *International Conference on Integrated Formal Methods*, pages 50–66. Springer, 2017.

[132] Grigore Rosu and Klaus Havelund. Rewriting-Based Techniques for Runtime Verification. *Automated Software Engineering*, 12(2):151–197, 2005.

[133] Christopher A Rouff, Walt Truszkowski, James Rash, and Mike Hinchey. A Survey of Formal Methods for Intelligent Swarms. Technical report, SAIC and NASA, 2004.

[134] Christopher A Rouff, Amy Vanderbilt, Walter Truszkowski, James L Rash, and Michael G Hinchey. Formal Methods for Autonomic and Swarm-based Systems. In *International Symposium on Leveraging Applications of Formal Methods*, pages 100–102, 2004.

[135] Erol Şahin. Swarm Robotics: From Sources of Inspiration to Domains of Application. In Erol Şahin and William M Spears, editors, *Swarm Robot.*, pages 10–20, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[136] C. Schlegel and R. Worz. The software framework SMARTSOFT for implementing sensorimotor systems. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*, volume 3, pages 1610–1616. IEEE, 1999.

[137] Danbing Seto, Bruce Krogh, Lui Sha, and Alongkrit Chutinan. The simplex architecture for safe online control system upgrades. In *American Control Conference, 1998. Proceedings of the 1998*, volume 6, pages 3504–3508. IEEE, 1998.

[138] Azamat Shakhimardanov, Nico Hochgeschwender, and Gerhard K. Kraetzschmar. Component models in robotics software. In Elena Messina and Raj Madhavan, editors, *Proc. 10th Perform. Metrics Intell. Syst. Work. - Permis '10*, page 82, Baltimore, USA, 2010. ACM Press.

[139] R. Simmons, C. Pecheur, and G. Srinivasan. Towards automatic verification of autonomous systems. In Hiroshi Ishikawa, Kazuo Tanie, and Hideki Hashimoto, editors, *Int. Conf. Intell. Robot. Syst.*, volume 2, pages 1410–1415, Takamatsu, Japan, 2000. IEEE.

[140] Prasad A Sistla, Milos Zefran, and Yao Feng. Runtime monitoring of stochastic cyber-physical systems with hybrid state. In Sarfraz Khurshid and Koushik Sen, editors, *Runtime Verif.*, volume 7186 LNCS, pages 276–293, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[141] Graeme Smith and Qin Li. MAZE: An Extension of Object-Z for Multi-Agent Systems. In Y Ait Ameur and KD Schewe, editors, *Int. Conf. Abstr. State Mach. Alloy. B, TLA, VDM, Z*, volume 8477 of *LNCS*, pages 72–85, Toulouse, France, 2014. Springer, Springer.

[142] Thierry Sotiropoulos, Hélène Helene Waeselynck, Jeremie Guiochet, and Felix Ingrand. Can robot navigation bugs be found in simulation? An exploratory study. In Mladen Vouk, Irena Bojanova, Manuel Nuñez, Tadashi Dohi, and Xiaoying Bai, editors, *Proc. - 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2017*, pages 150–159, Prague, Czech Republic, 2017. IEEE.

[143] Dennis Stampfer. The SmartMDSD Toolchain : An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software. *Journal of Software Engineering for Robotics*, 1(July):3–19, 2016.

[144] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merk. *Eclipse Modeling Framework*. Addison-Wesley Professional, Boston, USA, 2nd edition, 2008.

[145] Anton Tarasyuk, Inna Pereverzeva, Elena Troubitsyna, Timo Latvala, and Laura Nummila. Formal development and assessment of a reconfigurable on-board satellite system. In Frank Ortmeier and Peter Daniel, editors, *Proceedings of 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP 2012)*, volume 7612 of *Lecture Notes in Computer Science*, page 210222, Magdeburg, Germany, 2012. Springer-Verlag Berlin Heidelberg.

[146] Moritz Tenorth and Michael Beetz. Knowrobknowledge processing for autonomous personal robots. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4261–4266. IEEE, 2009.

[147] A. Tiwari. Abstractions for Hybrid Systems. *Formal Methods in Systems Design*, 32:57–83, 2008.

[148] Moshe Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. In *Proc. Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop)*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.

[149] Willem Visser, Klaus Havelund, Guillaume P. Brat, Seungjoon Park, and Flavio Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.

[150] Mirko Wächter, Simon Ottenhaus, Manfred Kröhnert, Nikolaus Vahrenkamp, and Tamim Asfour. The ArmarX Statechart Concept: Graphical Programing of Robot Behavior. *Frontiers in Robotics and AI*, 3(June):33, 2016.

[151] Dennis Walter, Holger Täubig, and Christoph Lüth. Experiences in applying formal verification in robotics. In Erwin Schoitsch, editor, *Computer Safety, Reliability, and Security*, pages 347–360, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[152] M. Webster, N. Cameron, M. Fisher, and M. Jump. Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. *Journal of Aerospace Information Systems*, 11(5):1–31, 2014.

[153] Matt Webster, Clare Dixon, Michael Fisher, Maha Salem, Joe Saunders, Kheng Lee Koay, and Kerstin Dautenhahn. Formal verification of an autonomous personal robotic assistant. In *AAAI Spring Symposium Series*, pages 74–79, Palo Alto, USA, 2014. AAAI.

[154] Matt Webster, Clare Dixon, Michael Fisher, Maha Salem, Joe Saunders, Kheng Lee Koay, Kerstin Dautenhahn, and Joan Saez-Pons. Toward Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study. *IEEE Transactions on Human-Machine Systems*, 46(2):186–196, apr 2016.

[155] Matt Webster, Michael Fisher, Neil Cameron, and Mike Jump. Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems. In F Flammini, S Bologna, and V Vittorini, editors, *Comput. Safety, Reliab. Secur.*, volume 6894 of *LNCS*, pages 228–242, Naples, Italy, 2011. Springer, Berlin, Heidelberg.

[156] Danny Weyns, M Usman Iftikhar, Didac Gil de la Iglesia, Tanvir Ahmad, Didac Gil, De Iglesia, and Tanvir Ahmad. A Survey of Formal Methods in Self-adaptive Systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 67–79, New York, NY, USA, 2012. ACM.

[157] Danny Weyns, Sam Malek, and Jesper Andersson. Forms: a FOrmal Reference Model for Self-adaptation. In *International conference on Autonomic computing*, page 205, New York, New York, USA, 2010. ACM.

[158] Danny Weyns, Sam Malek, and Jesper Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(1):8, 2012.

[159] Alan F T Winfield, Wenguo Liu, Julien Nembrini, and Alcherio Martinoli. Modelling a wireless connected swarm of mobile robots. *Swarm Intell.*, 2(2-4):241–266, dec 2008.

[160] Alan F.T. Winfield and Julien Nembrini. Safety in numbers: fault-tolerance in robot swarms. *Int. J. Model. Identif. Control*, 1(1):30, 2006.

[161] Alan FT Winfield, Jin Sa, Mari-Carmen Fernndez-Gago, Clare Dixon, and Michael Fisher. On Formal Specification of Emergent Behaviours in Swarm Robotic Systems. *International Journal of Advanced Robotic Systems*, 2(4):39, 2005.

[162] V. A. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, and H Costelha. Petri Net Plans: A Formal Model for Representation and Execution of Multi-robot Plans. In L Padgham, editor, *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, volume 23 of *AAMAS '08*, pages 79–86, Estoril; Portugal, nov 2008. International Foundation for Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems.