Formal Specification and Verification of Autonomous Robotic Systems: A Survey

MATT LUCKCUCK, MARIE FARRELL, LOUISE A DENNIS, CLARE DIXON, and MICHAEL FISHER, Department of Computer Science, University of Liverpool, UK

Autonomous robotic systems are complex, hybrid, and often safety-critical; this makes their formal specification and verification uniquely challenging. Though commonly used, testing and simulation alone are insufficient to ensure the correctness of, or provide sufficient evidence for the certification of, autonomous robotics. Formal methods for autonomous robotics has received some attention in the literature, but no resource provides a current overview. This paper systematically surveys the state-of-the-art in formal specification and verification for autonomous robotics. Specially, it identifies and categorises the challenges posed by, the formalisms aimed at, and the formal approaches for the specification and verification of autonomous robotics.

ACKNOWLEDGMENTS

1 INTRODUCTION, METHODOLOGY AND RELATED WORK

An *autonomous system* is an artificially intelligent entity that makes decisions in response to input, independent of human interaction. *Robotic systems* are physical entities that interact with the physical world. Thus, we consider an *autonomous robotic system* as a machine that uses Artificial Intelligence (AI), has a physical presence in and interacts with the real world. They are complex, inherently hybrid, systems, combining both hardware and software; they often require close safety,

Authors' address: Matt Luckcuck, m.luckcuck@liverpool.ac.uk; Marie Farrell, marie.farrell@liverpool.ac.uk; Louise A Dennis, l.a.dennis@liverpool.ac.uk; Clare Dixon, cldixon@liverpool.ac.uk; Michael Fisher, mfisher@liverpool.ac.uk, Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK.

legal, and ethical consideration. Autonomous robotics are increasingly being used in commonplace-scenarios, such as driverless cars [68], pilotless aircraft [176], and domestic assistants [60, 174].

While for many engineered systems, testing, either through real deployment or via simulation, is deemed sufficient; the unique challenges of autonomous robotics, their dependence on sophisticated software control and decision-making, and their increasing deployment in safety-critical scenarios, require a stronger form of verification. This leads us towards using formal methods, which are mathematically-based techniques for the specification and verification of software systems, to ensure the correctness of, and provide sufficient evidence for the certification of, robotic systems.

We contribute an overview and analysis of the state-of-the-art in formal specification and verification of autonomous robotics. §1.1 outlines the scope, research questions and search criteria for our survey. §1.2 describes related work concerning formal methods for robotics and differentiates them from our work. We recognise the important role that middleware architectures and, non- and semi-formal techniques have in the development of reliable robotics and we briefly summarise some of these techniques in §2. The specification and verification challenges raised by autonomous robotic systems are discussed next: §3 describes the challenges of their context (the *external* challenges) and §4 describes the challenges of their organisation (the *internal* challenges). §5 discusses the formalisms used in the literature for specification and verification of autonomous robotics. §6 characterises the approaches to formal specification and verification of autonomous robotics found in the literature. Finally, §7 discusses the results of our survey and presents our observations on the future directions of formal methods for autonomous robotic systems.

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 analysed, and outline the search criteria that were followed.

Scope: Our primary concern is the formal 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 that 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 humans but also those that can be controlled by humans, though we do not model human behaviour in detail as this is beyond the current capabilities of formal methods.
- 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, including safety, security, reliability, efficiency, etc., primarily concerning the functional behaviour of autonomous robotic systems software. In particular, we do not consider mechanical, materials, or physics issues since our focus is on the software that is controlling these systems.
- By 'formalisms' we mean mathematically-based techniques for software and systems development. We do not cover techniques based on, for example, systems of differential equations except to mention them as part of hybrid approaches.

Ref	Formalisms/Tools Employed	Case Study or Example Used
[174]	SPIN, Brahms modelling language (BrahmsToPromela translator), LTL.	Care-O-Bot robotic assistant in a Robot House Environment.
[71]	GWENDOLEN, AJPF, MCAPL, LTL.	RoboCup rescue scenario, autonomous satellite scenario, autonomous pilotless aircraft scenario.
[113]	Quantified differential dynamic logic (QdL) and proof using KeYmaeraD.	Formally verified control algorithms for surgical robots.
[12]	Reachability analysis.	A wrong-way driver threatens two autonomously driving vehicles on a road with three lanes.
[110]	PCTL models of Probabilistic State Machines, PRISM.	The foraging (swarm) robot scenario [118].
[132]	Process Algebra for Robotic Systems (PARS) and MissionLab.	Search for a Biohazard.
[178]	Z model for self-adaptive systems, CZT.	MAPE-K reference model [105] and Layered self-adaptive robotics software [65].
[125]	KeYmaera, dynamic differential logic for hybrid systems.	Robotic ground vehicle navigation.
[176]	GWENDOLEN, SPIN, AJPF, LTL.	Model checking for certification of autonomous pilotless aircraft systems.
[20]	BIP (Behaviour-Interaction-Priority) component framework, Evaluator model-checker (temporal logic properties) and monitors.	DALA (iRobot ATRV).
[71]	Theoretical work inspired by LTL and model-checking, computational framework where task specifications are regular expressions. Finite State Automata.	Car-like robots operating in an urban-like environment.
[138]	Quartz (synchronous language),model-checking LTL/CTL specifications using Beryl model-checker in the Averest verification framework.	Behaviour-based control network of mobile outdoor robot RAVON.
[43]	A metamodel-based translation from AADL to BIP. Aldebaran model-checker (in BIP) for deadlock detection and observers for monitoring safety properties.	Flight computer.
[133]	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.
[50]	SPIN to verify Ach (novel interprocess (IPC) communication mechanism and library).	Implementation of Ach on Golem Kang humanoid robot.
[51]	Timed Automata and Timed CTL (TCTL) to verify MAPE-K templates for the development of self adaptive systems, and UPPAAL.	Traffic monitoring, outdoor learning robot, transportation and, storytelling robot.
[129]	Hybrid system modelling using natural language programming (sEnglish) which is translated to Stateflow and verified using MCMAS.	Autonomous underwater vehicles.
[30]	(Extended Abstract) Brahms modelling framework translated into Jason agent code and model-checking temporal logic properties.	Astronaut-robot collaboration.
[88]	Systems of differential equations and manoeuvrer automata generated using reachability analysis.	Collision avoidance of road vehicles.
[141]	LTL, extension to LTLMoP toolkit for robot mission planning.	Fire fighting scenario.
[24]	BIP framework, DFinder model-checker.	DALA autonomous rover.
[146]	Extensible Agent Behaviour Specification Language (XABSL) - language for describing state machines.	Robot Soccer.
[138]	Synchronous language Quartz and verification using model-checking techniques of the Averest verification framework (Beryl).	Mobile outdoor robot RAVON.
[97]	Timed Automata (TA) for modelling and Timed Computation Tree Logic (TCTL), for safety and correctness properties for an adaptive autonomous system.	Distributed network of traffic monitoring cameras that reorganise to coordinate data transmission.
[66]	RV-BIP, a tool that produces runtime monitors for robots the BIP framework from formal descriptions of the system and monitor.	Autonomous rover.
[98]	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.
[182]	Probabilistic Finite-State Machine (PFSM), transition probabilities are estimated and then validated by simulation in Player/Stage.	Swarm navigating with the alpha algorithm.
[49]	Layered modelling of a robot's task plans using Petri Nets.	Robot Soccer.
[179]	Reference model for building verifiable self-adaptive systems (FORMS).	Traffic monitoring system.
[39]	Modelling an abstract architecture for reactive multi-agent systems, which are then analysed for deadlock freedom.	Small example with two agents cooperating to move objects.
[186]	Modelling plans as Petri nets with an additional set of Goal Markings for single- and multi-robot systems.	Multi-robot foraging and object passing.
[123]	Bio-PEPA process algebra to model the 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.
[108]	LTL-X (LTL without "next" operator) and Büchi Automata to give a a robot control and communications strategy with reduced 'stop-and-wait' synchronisations.	A team of unicycle robots with distributed control.
[117]	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 [a].
[6]	Derives a specification of a real-time system (in CSP+T) from a UML-RT Model for design-time schedulability and dependability analysis.	Two-armed industrial robot and mechanical press.
[80]	Methodology for verifying three cooperative robots with distributed control using the formal language KLAIM, and its extension for stochastic analysis.	Three cooperative robots transporting an object to a goal area.
[11]	Ontology-based (Web Ontology Language (OWL)) reconfiguration agent in a manufacturing environment.	Four machines: (1) processing, (2) handling, (3) Simple Conveyor Chain, and (4) filling station.
[166]	KnowRob ontology which is based on description logic, OWL.	Household robot setting a table.
[137]	IEEE-RAS Ontology for robotics and automation.	N/A
[104]	Generating robot motion plans that satisfy properties in a deterministic subset of μ-calculus.	N/A
[116]	Generating robot behaviour automata that satisfy LTL specifications, uses Linear Temporal Logic MissiOn Planning (LTLMoP) and TuLiP.	Driverless car in an urban environment.
[70]	Distributed LTL specifications over multiple agents, exchanged when the robots physically meet and model checked for mutual satisfiability.	N/A
[164]	First-Order Logic (FOL) to formalise agent beliefs and intentions, and uses these to predict the plan of other agents for cooperation.	Willow Garage PR2 Robot, using Robot Operating System (ROS) in a human-robot team scenario.
[64]	Uses ReadyLog to formalise the basic concepts of football theory, in order to formally describe football tactics.	RoboCup football robots.
[29]	UML, Alloy state machines, KodKod SAT solver, Linear-time Temporal Logic (LTL) and FOL.	Surgical operations.
[115]	Fragment of temporal logic called General Reactivity (1).	Search and rescue style missions.
[18]	Behaviour Interaction Priority (BIP) framework.	DALA iROBOT ATRV.
[25]	BIP and LAAS frameworks.	DALA iROBOT ATRV.
[83]	Büchi Automata (BA) modelling the environment and LTL task specifications.	A surveillance robot.
[48]	Markov chains to predict swarm aggregation.	Cockroach aggregation.
[89]	Integration of Object-Z and Statecharts, SAL model checker, and STeP theorem prover.	N/A
[180]	Requirements language for dynamically adaptive systems (RELAX) whose semantics is defined in terms of temporal fuzzy logic.	Adaptive assisted living smart home.
[81]	Restore Invariant Approach (RIA), Microsoft Robotics Studio simulator, JADEX multi-agent framework, Alloy, class diagrams and OCL.	Adaptive production cell.
[55]	LTL and Agent Java PathFinder (AJPF).	Three pilotless civilian aircraft examples: brake failure, erratic intruder aircraft, and low fuel.
[57]	LTL and AJPF.	Ethical scenario of a robot trying to stop humans from falling down a hole.
[95]	LTL.	LandShark rover robot.
[1]	BIP models to generate G^realM controllers, and D-Finder tool.	Dala rover navigating to a target position, taking a picture, and returning to the initial position.
[111]	Discrete-Time Markov Chains (DTMCs) to model the foraging algorithm, Probabilistic Computation Tree Logic (PCTL) for safety properties, PRISM.	Foraging robot swarm.
[36]	DTMCs to model the aggregation algorithm, Probabilistic Computation Tree Logic* (PCTL*) specifications, and PRISM for checking.	Aggregation in robot swarm.
[61]	Finite-State Machines (FSMs), Probabilistic Temporal Logic (PTL) for specification, checked in NuSMV.	Alpha algorithm for robot swarms.
[62]	FSMs, PTL for specification, checked in NuSMV. Also a NetLogo simulation.	Alpha algorithm for robot swarms.
[41]	An update of [40] that uses JFLAP and MatLab.	Robotic urban-like environment (RULE).
[175]	BrahmsToPromela and SPIN model-checker.	Care-O-Bot.

Table 1. These are the papers that were obtained using the methodology outlined in §1.1 that were deemed to be in scope.

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

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

RQ2: What are the current formalisms, tools, and approaches used when 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?

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. We answer **RQ1** in §3 and §4 where we identify the current *external* and *internal* challenges, respectively, to formally specifying and verifying autonomous robotic systems that we have derived from our examination of the literature. We provide a thorough analysis of our findings in §5 and §6, where we examine potential answers to **RQ2** and **RQ3**. §7 provides a thorough discussion of how we answer these research questions and our observations of the literature.

Search Criteria: Our search queries were *formal modelling of (autonomous) robotic systems*, *formal specification of (autonomous) robotic systems*, and *formal verification of (autonomous) robotic systems*. Our search travelled 5 pages deep of Google Scholar (on 21/05/2018) and discounted any results that were obviously out of scope. In total, we surveyed 156 papers of which 63 were deemed to be in scope and these are described in Table 1. We restricted our search to papers that were published in the last ten years (2007 – 2018). Note that we include those that were published up until the date the search was carried out in 2018. To contextualise these results we did some 'snowballing' from the core set of 63 papers and these will be mentioned throughout this survey, however, our analysis specifically focuses on this core set in order to preserve the integrity of our results.

We have broadly followed the methodology in [107] by delineating research questions, scope and search criteria. However, unlike the usual systematic review, we have employed 'snowballing' to provide a more rounded view of the topic and to minimise omissions. Generally, systematic reviews provide a detailed enumeration of the tools or approaches used in a specific domain. They do not normally, however, describe or discuss these tools or approaches in great detail. We have crafted this survey to not only summarise and enumerate the current state-of-the-art, but also to educate the reader and provide a point of reference for researchers in this area.

1.2 Related Work

To our knowledge, there is no survey of the recent literature for formal specification and verification of autonomous robotic systems. This section differentiates related work from ours.

Robotic systems often have safety-critical consequences. A survey on safety-critical robotics [82] 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. In contrast, our survey focusses on the application of formal methods to any type of autonomous robotic system.

Nordmann et al. [131] present a detailed survey of Domain Specific Modelling Languages for robotics. Their study focusses on non-formal modelling, but was influential to the depth and organisation of our survey. 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.

In applying formal verification methods to programming languages for autonomous software, Simmons et al. [156] identified three challenges for the automatic verification of autonomous systems: (1) "special purpose languages use special-purpose interpreters or compilers" that must be verified, (2) "application-specific programs written in these languages need to be verified for internal correctness" (safety, liveness, etc.) and (3) "programs need to be verified for external correctness" (how they interact with the overall software system).

Weyns et al. [177] present a survey 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 and so it is somewhat tangential to our survey. We discuss the specific challenges of formally specifying and verifying self-adaptive robotic systems in §4.3.

Rouff et al. [149, 150] survey formal methods for intelligent swarms. As we discuss in §4.2.1, the challenges posed by robot swarms are an extension of those of single autonomous robotic systems. They compare a wide range of formalisms, covering process algebras, model-based notations, various logics, amongst others. They conclude that no single formalism is suitable to the range of challenges posed by intelligent robot swarms. Their focus is, again, narrower than ours and §4.2.1 offers a more recent perspective on formal methods for robot swarms.

2   GENERAL SOFTWARE ENGINEERING TECHNIQUES FOR ROBOTIC SYSTEMS

The complex task of engineering a reliable robotic system is often addressed using a variety of 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 and complete view of robotic system engineering, beginning with the commonly used middleware architectures.

Middleware Architectures: Architectures for developing robotic systems generally adopt the paradigm of communicating *components*/*nodes*. Among the most popular in the literature are: ROS [140], OPRoS [100], OpenRTM [13], Orocos [151], and G^{en}oM [73]. Each of these supports engineering a robotic system in a slightly different way but most share a common set of component-based, modular concepts [155] to facilitate reuse of modules between different frameworks. Supporting the integration of middleware frameworks, [5] presents a robotic software manager with a high-level API that bridges the gaps between ROS, OPRoS, and OpenRTM.

It is often assumed that a middleware is sound, and this is key to trusting the robotic system [75]. However, given the heterogeneity of the systems that can be produced using such architectures and their parametrisable nature, guaranteeing correct robot behaviour is challenging [86]. Thus, we have identified the following categories of non- and semi-formal methods.

Testing and Simulation: Both testing and simulation are useful when developing robotic systems. Particularly simulation, which enables the examination of the system's behaviour in statistically unlikely situations. However, they are time-consuming, and examine part of the program's state space, so they cannot be used to reason reliably about properties of the whole program. Moreover, field tests are potentially dangerous to life and the robot hardware Examples include the following uses of field tests [75, 86, 110, 128] and/or simulations [9, 110].

Domain Specific Languages (DSLs): An extensive literature survey identified 137 state-of-the-art DSLs for robotics [131], the majority of which were for capturing robotics architectures and for use within mature subdomains (e.g. robot motion). Conversely, fewer DSLs were aimed at emerging

subdomains (e.g. force control). The majority of surveyed DSLs 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 [162] meta-models. Related work on providing a robotics-specific Integrated Development Environment (IDE) [161] aims to integrate the various phases of robotics software development by guiding the user through a standardised development process. The Declarative Robot Safety language (DeRoS) [4] is a DSL for describing a robot's safety rules and corresponding corrective actions. Usefully, this provides automatic code generation to integrate these rules with runtime monitoring by generating a ROS safety monitoring node.

Graphical Notations: Graphical Notations are present and examined throughout the literature. They usually aim to provide an easy to use mechanism for designing robotic software that enables communication between engineers from the various distinct disciplines involved in robotics [87, 124, 171]. Statecharts [87] form the basis for several of these graphical notations, such as ArmarX statecharts [171] which are executable as C++ programs. A similar executable notation is *restricted Finite State Machine* (rFSM) Statecharts [124], which represent a subset of UML 2 and Statecharts [87]. Other notations include RoboFlow [10] which adopts the style of flow charts. RoboFlow is specifically designed for engineering robot movement and manipulation tasks.

MDE/XML:. Numerous Model-Driven Engineering (MDE) approaches use the Eclipse Modelling Framework's Ecore meta-meta-model [72]. The work in [94] presents a mapping from the XML-based AutomationML into ROS program code. The BRICS Component Model [38] translates high-level models into platform-specific component-models for Orocos and ROS. Another model-driven approach is the MontiArcAutomaton architecture description language [145] which translates models into program code, mostly for ROS [3]. Performance Level Profiles [33] is an 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 the output from their tool is flexible and not tied to ROS. RADAR is a novel procedural language, with a Python implementation, for describing robot event responses [28].

Discussion: Middleware frameworks provide a flexible standard interface to the robotic hardware and the above non- and semi-formal approaches aim to adapt the parts of software engineering practice that are relevant to the robotics domain. DSLs abstractly describe properties about specific parts of the system, often improving the understanding or interoperability of the system. Graphical notations aid in communication and visualisation. However, more formal languages provide unambiguous semantics and enable reasoning about the whole program rather than only a particular set of program behaviours. Next, we describe the *external* challenges faced when applying formal methods to robotic systems. These challenges are external to the robotic system and are independent of the type of robotic system being developed.

3 EXTERNAL CHALLENGES OF AUTONOMOUS ROBOTIC SYSTEMS

This section discusses the inherent challenges of formally specifying and verifying any autonomous robotic system. We categorise them as *external* challenges because they are independent of the robotic system's internal design.

§3.1 discusses the challenge of modelling a robotic system's physical environment and verifying its behaviour within that environment. This process is of paramount importance because real-world interactions can significantly impact safety.

§3.2 discusses the challenge of providing sufficient evidence for both certification and public trust. For some robotic systems, the domain in which they are deployed has a certification body (for

example, in the nuclear industry) that describes the evidence required to certify the system for use. Other robotic systems are being used in domains where there is no certification body (domestic assistants, for example) so evidence must be provided to gain trust in their safety. Both situations are challenging because of the technical nature of both robotic systems and formal verification.

3.1 Modelling the Physical Environment

As mentioned in §1, we consider an autonomous robot as a machine that implements Artificial Intelligence (AI) techniques, has a physical presence, and interacts with the world. Thus, one of the most prominent challenges in verifying robotic systems is its interaction with an unstructured environment. This is further complicated by differing sensor accuracy or motor performance, or components that are degraded or damaged.

Interactions between a robot and its physical environment can have a major influence on its behaviour, because the robot may be reacting to environmental conditions not considered at design-time. Indeed, the behaviour of adaptive systems exhibit is directly driven by environmental interactions. While it is accepted that formally modelling a robotic system *within* its physical environment is important [71, 172], it has had limited attention in the literature. We believe that this is due to the sheer complexity of this task: aside from the difficulty of modelling the real world, a robot only has partial knowledge of its surroundings. Sensing limitations caused by sensor blind-spots and interference between sensors add extra complexity to the modelling process [31, 88, 134].

To address the challenge of combining discrete computations and a continuous environment, a robotic system is typically separated into several layers [8, 71]. 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 [140] or G^{en}oM [73]) 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.

Some previous work has focussed on a robot's decision making, ignoring its environment [110, 120]. Others assume that the environment is static and known, prior to the robot's deployment [76, 128, 174], which is often neither possible nor feasible [88]. For example, the environment may contain both fixed and mobile objects whose future behaviour is unknown [31] or the robot's goal may be to map the environment, so the layout is unknown. Hybrid architectures often take a third approach; to abstract away from the environment and assume that a component has the ability to provide predicates that represent the continuous sensor data about the robot's environment [55, 57, 71, 163, 168]. While this encapsulates the high-level control, insulating it from the environment, the implicit assumption of a static, known environment can often remain.

Using models of the environment that are static and assume prior knowledge may limit their effectiveness. However, Sotiropoulos et al. [160] examine the ability of low-fidelity environmental simulations to reproduce bugs that were found during field tests of the same robot. 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.

Probabilistic models are a popular approach to capturing dynamic and uncertain environments. In [130], 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.

Similarly, Hoffmann et al. [93] formalise a pilotless aircraft and its physical environment using an extension of Probabilistic Finite-State Machines (PFSMs). They use PRISM to verify properties about their model, with the pilotless aircraft tasked with foraging for objects, which it must return to a drop-off location. Their approach involves running small-scale simulations of the pilotless aircraft in its environment to determine the probabilities and timing values for their formal model. In contrast, Costelha and Lima [49] 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 with the environment model using untimed direct transitions and other layers in the models using stochastic Petri Nets.

Modelling the environment is particularly relevant for navigation, and tackling collision avoidance and safe robot navigation [31, 125, 134, 138] often feature in the literature. Although several navigation algorithms exist, some cannot be used in safety-critical scenarios because they have not been verified [134]. The Simplex architecture [154] provides a gateway to using these unverified algorithms with the concept of advanced controllers (AC). ACs are used alongside a verified baseline controller (BC) and a decision module, which chooses which controller is active. The decision module monitors the system and if it determines that the AC will violate one of the safety properties, then the BC takes control. Other work employs reachability analysis to generate a manoeuver automaton for collision avoidance of road vehicles [88]. Here, differential equations model the continuous behaviour of the system. Runtime fault monitoring is useful for catching irregular behaviours in running systems but it does not ensure safety in all situations.

Mitsch et al. [125] use differential dynamic logic (dL) [135], 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. verify that the robot will not collide with, and maintains a sufficient distance from, stationary and moving obstacles. In proving these safety properties, 85% of the proof steps were carried out automatically.

In further work, Mitsch et al. [126] verify a safety property that makes less conservative driving assumptions, allowing for imperfect sensors; and add liveness proofs, to guarantee progress. 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 [71, 172]. This is especially 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 is often intractable [58]. Moreover, in a multi-robot setting one robot can be part of another's environment [115]. 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 [172].

3.2 Evidence for Certification and Trust

Robotic systems are frequently deployed in safety-critical domains, such as 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, thus, essential. However, autonomous robotic systems development often falls short of providing sufficient evidence, or evidence in the correct form, for these certification bodies [75].

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 for such a pilotless aircraft are still under development [173]. 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 [112]. In fact, one of the main difficulties here is in choosing a formalism so that a good compromise is reached between how expressive the formalism is and the complexity of its analysis/synthesis algorithms [22]. This also presents a challenge for certification bodies who need to be able to determine the reliability of safety-critical robotic systems and the formal methods that were employed during development.

Autonomous vehicles are usually regulated by the relevant vehicle authority for the territory that they are used in. However, their introduction also requires public trust. The majority of respondents of a recent survey on public attitudes towards driverless cars (UK, USA, and Australia) had both a high expectation of their benefits and a high level of concern over their safety and security [152].

In contrast to the nuclear and aerospace domains, robots used in domestic or care environments do not have strong certification requirements but 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, unsafe or unfriendly [60]. This lack of both trust and safety assurances can hamper adoption of robotic systems in wider society [76], even where they could be extremely useful and potentially improve the quality of human life.

Isabelle/HOL and temporal logic have been used to formalise a subset of traffic rules for vehicle overtaking in Germany [147]. 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 in order to not be considered to blame for any accident [125]. In related work, Webster et al. [176] utilise model-checking to provide certification evidence for an autonomous pilotless aircraft system using the Gwendolen agent language and the Agent Java PathFinder (AJPF) approach as an improvement over Promela and SPIN.

The certification process often requires safety cases which provide a structured argument of a system's safety in a certain context, given certain assumptions. Automating safety case generation is challenging; safety cases must combine elements of the physical design, software design, and the maintenance of both. Denney and Pai [52] have described a methodology for automatically generating safety cases for the Swift pilotless aircraft system using a domain theory and their AutoCert tool [53], which automates the generation of safety case fragments.

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 [16, 17, 37]. Worryingly, they ignore the potential situation where no ethical choice is available, and there seems to be little consideration of this situation in the literature [55]. Clearly, reasoning about ethical behaviour is at least as difficult with robots as it is with humans.

3.3 Summary of External Challenges

This section identified two main external challenges to the specification and verification of robotic systems in the literature. External challenges come from the operating and design environment of the robotic system. The challenges 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 [134]. 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 (both computationally and in terms of usability) of formal models, particularly if we consider the modelling of a complex physical environment.

Formal techniques can be used to provide trust and certification evidence, but it is clear that current techniques are insufficient. This is further complicated by the small number of available guidelines to help developers to identify the most appropriate formal method(s) to specify and verify their system [112]. Regulators are more concerned with the evidence of safety, than with the specific methods used to produce the evidence; therefore, determining suitable and robust formal methods for distinct robotic systems remains an open problem.

These are the challenges, *external* to the robotic system, that our survey identified. Next, we discuss the *internal* challenges, which stem from how a robotic system is engineered.

4 INTERNAL CHALLENGES OF AUTONOMOUS ROBOTIC SYSTEMS

Autonomous robotic systems are increasingly prevalent, and are attractive for removing humans from hazardous environments such as nuclear waste disposal or space exploration [77]. Our survey identified agent-based approaches as a popular way of providing the autonomy that these systems need; we discuss the challenges that this approach brings to formal specification and verification in §4.1. Multi-robot systems are also an active area of research; §4.2.1 and §4.2.2 describe the challenges of formally specifying and verifying homogeneous and heterogeneous multi-robot systems, respectively. In hazardous environments, autonomous robotic systems need to be capable of adaptation and reconfiguration in response to changes in the environment, system, or mission; §4.3 discusses the challenges posed by these systems.

4.1 Agent-Based Systems

An *agent* encapsulates the system's decision-making capability into one component. This helps to provide *rational* autonomy (where a system can explain its reasoning) which is crucial for providing evidence to gain public trust or for certification (discussed in §3.2). Since the implementation of Procedural Reasoning System (PRS) [78], Belief-Desire-Intention (BDI) agents received a great deal of attention [142]. BDI systems can use a single or multiple agents, depending on their requirements. The BDI model of agency appears to be prevalent in the surveyed literature.

The distributed Multi-Agent Reasoning System (dMARS) is an implementation of PRS with the BDI model operationalised by plans [59]. A formal Z specification of dMARS provides a formal foundation for the semantics of dMARS agents [59]. 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 [59].

Webster et al. [173, 176] verify an autonomous pilotless aircraft using a program model-checker. The aircraft is controlled by a BDI agent, written in the GWENDOLEN language [54], and verified by the AJPF model-checker [56]. The verified agent is plugged into a flight simulator [173] to visualise the verified scenarios. The work in [101, 102] describes a GWENDOLEN agent controlling a driverless car in a vehicle platoon. AJPF is used to verify safety properties related to the car joining and leaving a platoon, and for maintaining a safe distance during platooning. Verification of the timing properties is handled by Uppaal.

Similarly, Choi et al. model a system of nine heterogeneous agents with one representing the robot's physical environment [44]. They used the Model Checker for Multi-Agent Systems (MCMAS)

[119] 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 [129]. They use natural language programming (sEnglish) which results in an ontology that is used to represent a labelled transition system. This is then translated into Stateflow and verified with MCMAS. Interestingly, they represent the human interface as an agent, as does [30]. This links human behaviour and the formal model that they have developed.

Hoffmann et al. characterise the autonomous behaviour of an agent (based on the BDI model) as a PFSM extended with a weight function to map weights onto actions, which they call an *Autonomous Probabilistic Finite-State Machine* (APFSM) [93]. Due to 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. [32] propose the Autonomous Reasoning Engine (ARE) for controlling spacecraft, it provides plan generation, validation, execution, monitoring and fault-detection. It communicates with the ground station, performs autonomous reasoning, receives sensor data, and sends commands to actuators. A prototype ARE, developed using NuSMV, has been used in ESA rover and satellite projects to provide a model-based approach to on-board autonomy.

Podorozhny et al. [136] use Alloy to verify multi-agent negotiations. Their work enables properties of agent coordination and interaction to be checked, and properties of complex data structures (which may be shared between agents). Their work centres on a system where multiple agents cooperate to reach and negotiate to optimise achieving their common goal(s). They model agents' negotiations as Finite-State Machines (FSMs), which are then translated into an Alloy specification and verified using Alloy Analyzer. The main challenge was ensuring tractability of the Alloy models.

4.2 Multi-Robot Systems

Robotic systems composed of several robots present challenges that must be tackled in addition to those faced when developing single-robot systems. Two broad categories of multi-robot systems exist: *swarms*, where all of the robots exhibit the same behaviour; and *teams*, where the robots may each behave distinctly. In both cases, the behavioural requirements of the system must be considered at both the microscopic level (individual robots) and at the macroscopic level (whole system). Multi-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 [7].

The work in [80] presents a methodology for verifying cooperative robots with distributed control. They use the formal language KLAIM, which was designed for distributed systems; its stochastic extension, STOKLAIM; and its related tool set. They formalise a specification of the robot's behaviour and physical environment, in KLAIM; and then add stochastic execution times for actions, in STOKLAIM. They validate their modelling approach by comparing their results to those obtained from physics-based simulations of the same system. The case study contains three homogeneous robots, cooperating under distributed control, to transport objects. In this system, the robots have homogeneous behaviour, but are too few in number to be considered a swarm.

One approach to easing multi-robot system development formally relates the languages used at the macro- and micro-levels of abstraction. To this end, Smith and Li [159] present MAZE, an extension of Object-Z for multi-agent systems, which includes Back's action refinement. They describe a top-down refinement-based development process and several shorthand notations to improve robot specification at the micro-level.

Next, we discuss the specific challenges faced when developing homogeneous robot systems, specifically swarms, (in §4.2.1) and heterogeneous robot systems, specifically teams, (in §4.2.2).

4.2.1   Swarms. A robot swarm is a decentralised collection of robots that work together [23], often taking inspiration from swarms of ants or bees. Robot swarms generally contain robots that are: adaptive, large in number, not grouped together, relatively incapable or inefficient, and able to send and communicate locally [187]. However, these criteria are not exhaustive, and should simply be used to decide to what degree the term "swarm robotics" might apply [63].

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 [110]. However, the large number of robots in a swarm can explode the state space and thus hinder verification using conventional model-checking approaches [110]. A further issue is scalability: if you prove a property for a swarm of n robots, does the property hold for $n \pm 1$ robots?

Robot swarms are often used for their resilience. Winfield and Nembrini [183] examined the fault tolerance of 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 the *partial* failure of a robot was more problematic than a robot's total failure. In their example, a robot that was fully functioning, apart from its wheels not turning, anchored the swarm in place.

Determining if 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 is an active area of research [114]. Kouvaros and Lomuscio [114] present an approach that uses a temporal logic and fault injection to determine the ratio of faulty to functioning robots in a swarm navigating using the alpha algorithm. They specify temporal-epistemic properties and then they 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. The developers 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 [35, 120, 128]. 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 [120]. Winfield et al. [182] present a straightforward approach to 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[1].

Markov chains have been used to model the probabilities of members of a robot swarm joining or leaving aggregates of robots [48]. The probabilities were a function of the size of the aggregate, and were based on a biological study of cockroach aggregation. The model was used to predict the probability of a swarm aggregating, without capturing spatial information. Despite the limitation of assuming that an aggregate can only grow or shrink linearly, they conclude that robots need a minimal speed or sensor range to enable aggregation.

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 [150]. They concluded 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 for robot swarms, and more generally, software for robotic systems [92]. For example, the NASA Remote Agent uses specialized languages for each of the

---

[1]http://playerstage.sourceforge.net

planner, executive and fault diagnosis components [156]. We explore the use of integrated formal methods for the specification and verification of robotic systems in §6.4.

4.2.2 Teams. A robot team is similar to a swarm, but contains robots with different capabilities or behaviours to one another. For example, a search and rescue robot team might contain an autonomous aircraft to search the area, a wheeled robot to carry equipment, and a robot with caterpillar tracks that is capable of moving obstacles. The heterogeneity of the robot's capabilities and behaviour increases the likelihood that they will each need different formal methods during verification. Linking verification at the micro-level to the macro-level is, therefore, challenging. We discuss the use of integrated formal methods in more detail in §6.4.

The work in [108] presents a methodology for automating the development of robot teams, using a version of Linear Time Logic without the 'next' operator (LTL-X). They model-check a transition system describing the behaviour of the robot team for execution traces that satisfy an LTL-X formula. This trace is mapped to the communication and control strategy for each robot. Büchi Automata (BA) then produce a communication strategy that reduces the number of 'stop-and-wait' synchronisations that the team has to perform.

Robot teams often include robots with different capabilities, which produce different patterns of interaction. The work in [89] captures part of the Satisfaction-Altruism Model [157] to provide a formal model of the different and changing roles that different agents in a team can play. Each role defines a pattern of interaction. They propose using their model for specification and verification of a robotic team, and eventual refinement to code. Their model is formalised using an integrated formal method (discussed in §6.4) that combines Object-Z and Statecharts [87].

Previous work in this area has assumed that a robot and a human are in close contact and that the robot can continually observe the human's behaviour, allowing it to predict the human's plan. Talamadupula et al. [164] describe a First-Order Logic (FOL) formalisation of beliefs and intentions that allows a robot to predict another agent's plan. They explore a human-robot cooperation task where the robot must predict a human's plan based on mutually understood beliefs and intentions. The FOL predicates representing the beliefs and intentions of another agent (human or robotic) are input to the robot's planning system, which predicts the other agent's plan. An extension caters to incomplete knowledge of the other agent's goals.

4.3 Self-Adaptive and Reconfigurable Systems

Ensuring fault tolerance of robotic systems that operate in hazardous environments, potentially isolated from human interaction, is crucial. These systems must be able to diagnose and to reconfigure themselves in order to adapt to changes in their requirements or operating environment. Reconfiguration can help maintain quality of service or meet functional objectives [178]. This is particularly relevant for robotic systems that are deployed in hostile environments, such as in space or the deep ocean [77, 165]. *Rational* autonomy, where a system can explain its reasoning, is crucial for safe reconfiguration. Therefore, modelling the motivations and decisions of the system is an important line of current work. Ideally, this should be done in a way that enables formal verification of the autonomous decisions [71]. This is discussed in more detail in §4.1.

A self-adaptive system continually alters its behaviour in an environmentally-driven feedback loop. The findings of a literature survey indicated that while there are no standard tools for the formal modelling and verification of self-adaptive systems; 40% of the studies they surveyed used formal modelling or validation tools, and 30% of those tools were model-checkers [177]. Architecture-based self-adaptive systems are a specific class of self-adaptive system where the system reasons about a model of itself and the environment, and 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 (MAPE-K) [51]. A series of MAPE-K templates has been formally verified using Timed Automata (TA) and Timed Computation Tree Logic (TCTL) to aid in development.

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 [165]. The reconfigurable systems literature focuses on two research questions: (1) how to specify and analyse a reconfigurable system [130], and (2) how to compare different configurations of the system [130, 178]? Therefore, it seems crucial that formal techniques applied to reconfigurable systems are able to tackle these questions.

The development of reconfigurable machines is an area 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 [27]. The development of such systems places a heavy emphasis on designing these systems to be modular, thus making reconfiguration a more approachable endeavour.

Reconfigurability can be acheieved by developing a flexible control system that can reconfigure itself when a fault is detected [34]. Specifying this can be challenging, however. The work in [98] 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 Virtual Machine (VM). Executing the TA models ensures that the model's verified behaviour is the program's runtime behaviour. Recent work includes a methodology for verifying reconfigurable timed net condition/event systems [84] and stochastic petri nets to verify reconfigurable manufacturing systems [167].

One avenue of research suggests providing (both semi-formal and formal) models to be used at runtime [42]. 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 has the benefit of providing online assurance of an adaptive system, *while* it is adapting; which can improve the trust in results regarding safety.

Decentralised systems can bring their own set of challenges for adaptation and reconfigurability. Iftikhar and Weyns [97] 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. They model this system using TA and describe invariants and correctness properties using TCTL. Related work utilised a Monte Carlo approach to probabilistically verify a system of cooperating agents (autonomous vehicles) that implement a policy-based collision avoidance algorithm [133].

4.4 Summary of Internal Challenges

This section identified three internal challenges to the specification and verification of robotic systems related to how the robotic system is designed and built. These challenges are: agent-based, multi-robot, and reconfigurable systems.

Agent-based systems are not the only model of autonomy, but they encapsulate the description of autonomous behaviour into one component, and can be used to model interactions with other actors and the environment. A *rational* agent can explain its reasoning, which can be helpful during verification and in tackling the challenge of providing sufficient evidence for a certification body or to gain public trust (recall §3.2). Ensuring that agents are verifiable is a focus in the literature.

Multi-robot systems – both homogeneous swarms and heterogeneous teams – provide an interesting challenge to robotic systems. They provide resilient architectures; but their decentralisation brings specific challenges for formal verification. Reconfigurable systems are key to dealing with

	System		Property	
Formalism	**References**	**Total**	**References**	**Total**
Set-Based	[81], [89], [117], [178], [179]	5		0
State-Transition	[1], [20], [18], [25], [36], [39], [40], [41], [48], [49], [61], [62], [66], [70], [83], [88], [97], [98], [51], [99], [104], [108], [110], [111], [115], [116], [129], [141], [146], [174], [175], [182], [186]	33		0
Logics		6		32
Temporal Logic		0	[29], [36], [57], [55], [61], [62], [70], [71], [83], [95], [97], [98], [51], [99], [104], [108], [110], [111], [115], [116], [123], [129], [138], [141], [176], [174], [175]	27
Dynamic Logic	[113], [125]	2	[113], [125]	2
Other Logics	[29], [64], [80], [164]	4	[64], [80], [164]	3
Process Algebra	[6], [123], [132]	3	[132]	1
Ontology	[11], [129], [137], [166]	4		0
Other	[57], [55], [71], [138], [176]	5	[1], [20], [18], [25], [40], [41], [66], [180]	8

Table 2. Summary of the types of formalisms found in the literature for specifying the system and the properties to be checked. Note: some references appear in one half of the table but not the other because they only describe the specification of the system (or vice versa). Logics have been further subdivided into Temporal, Dynamic, and Other Logics.

changing environments and mission goals. However, ensuring that they reconfigure themselves in a safe way is a challenge, and the key focus in the literature.

These are the internal challenges that we derived from the literature, but of course, there may be others. For example, machine learning systems are becoming more prevalent in autonomous robotic systems and are notoriously difficult to formally verify, although recent work has emerged [96, 153].

5 FORMALISMS FOR ROBOTIC SYSTEMS

This section discusses the formalisms found in the literature, following the methodology from §1.1, that have been used to specify or verify robotic systems. Table 2 summarises the categories of formalisms being used to specify both the system and the properties being checked[2]. We explicitly display the formalisms used to specify both the system and the properties because it illustrates the (well known) links between types of formalism (e.g. state-transition systems and temporal logics); it also highlights potential gaps in the literature or areas to focus on when producing tools.

We found that systems were normally specified as state-transition systems and the properties using a logic – typically, a temporal logic. The 'Other' category contains those which do not fit in the other categories but are not numerous enough to warrant their own. Examples include the RELAX [180]requirements language and the Quartz language used with the Averest framework [138].

Table 3 summarises the wide variety of formal analysis tools that were found using our methodology. Several studies use tools that specify properties in temporal logic; e.g. UPPAAL [14], PRISM [35], etc. The number of tools in use for temporal logic is commensurate with their prevalence as illustrated in Table 2.

In this section, we describe the formalisms used to specify and verify robotic systems. §5.1 discusses set-based formalisms. §5.2 describes approaches using automata. §5.3 describes approaches

---

[2]We note that these are not definitive categories, but represent common terms for each type of formalism.

Type of Tool	Tool	References	Total	Type Total
Model-Checkers	PRISM	[110], [123], [111], [36]	4	25
	NuSMV	[61], [62]	2	
	UPPAAL	[97], [98], [51]	3	
	SAL	[89]	1	
	SPIN	[174], [175], [50], [70], [176]	5	
	Beryl	[138], [139]	2	
	Aldebaran	[43]	1	
	Dfinder	[24], [18], [25], [1]	4	
	Unspecified	[40], [30], [104]	3	
Program Model Checkers	AJPF	[71], [55], [57], [176]	4	7
	MCMAS	[114], [44], [129]	3	
Theorem Provers	KeyMaera	[125], [113]	2	3
	SteP	[89]	1	
Others	Bio-PEPA Tool Suite	[123]	1	14
	TmeNET	[49]	1	
	TuLiP	[116]	1	
	LTLMoP	[116], [141]	2	
	Alloy	[29], [81]	2	
	Evaluator	[20]	1	
	minisat	[20]	1	
	MissionLab (VIPARS)	[132]	1	
	RV-BIP	[66]	1	
	Community Z Tools	[117],[178], [179]	3	

Table 3. Summary of the formal verification tools, and the type of tools, identified in the core set of 63 papers surveyed. Some approaches use multiple tools, often to verify different parts of a system with the most suitable tool chosen for each individual component of the system.

using logics (temporal, dynamic and others). §5.4 discusses process algebras. §5.5 outlines ontologies that have been used, and §5.6 describes other existing formalisms. Finally, §5.7 summarises.

5.1 Set-Based Formalisms

Set-Based formalisms (such as the B-Method and Z) specify a system using set-theoretic representation and manipulation of data. They are well suited to capturing complicated data structures, but usually only provide limited, if any, features for capturing behaviour.

As 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 [178], which provides a Z model that describes an arbitrary self-adaptive system. They suggest that this model could be used in conjunction with various existing tools to 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 for other types of analysis. The model 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.

Liang et al. [117] use the Jaza animator for Z specifications, combined with a Java debugger, to provide a runtime monitoring system. A Z specification describes the system's behaviour, which is animated and compared to the running program via 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 to the monitored program.

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

In [165], Event-B specifications are combined with probabilistic properties to derive reconfigurable architectures for an on-board satellite system. Event-B evolved from the B-Method and is more suitable to modelling the dynamic behaviour of a system than its predecessor [2]. The combination of these formalisms allows the models of reconfigurations to be checked using PRISM for both the derivation (via refinement) of the system from its specification, and the probabilistic assessment of their reliability and performance.

5.2 State-Transition Formalisms

Petri Nets, FSMs and Finite-State Automata (FSA) are approaches to specifying behaviour as a state-transition system. Formalisms for discrete-event systems include those capturing time (e.g. TA or Timed Petri Nets) and probabilistic transitions (e.g. PFSM or Generalised Stochastic Petri Net (GSPN)). State-transition systems specify behaviour during the design phase, which is checked for properties like deadlock freedom; or used as an input to a tool that usually checks them for properties described in another formal language (e.g. a temporal logic, see §5.3.1). There is a close link between varieties of temporal logic and varieties of FSA [169], and many works combine both.

Petri Nets have had some use in modelling robotic systems. The work in [39] uses them to capture the abstract architecture of multiple reactive agents and they are analysed for deadlock freedom. In [186], Ziparo et al. extend Petri Nets to capture robot plans. This 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 [49] use four layers of Petri Nets to model a robot's task plans, each layer describes a distinct element at a different level of abstraction. They use Marked Ordinary Petri Nets (MOPNs) to specify the robot's environment to enable a more realistic model. Then they capture the actions that 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 UPPAAL model-checker uses networks of TA to describe input models, which are checked against temporal logic properties. The work in [86] targets the ROS middleware framework, capturing the communication between nodes using TA. The low-level information (e.g. queue sizes and time outs) allows the verification of safety and liveness properties with UPPAAL. In [97], TA are used to model a decentralised traffic monitoring system, and TCTL to describe system invariants and correctness properties of the system's self-adaptive behaviour which are verified using UPPAAL.

The work in [121] 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) which will be discussed in §5.4.

As described in §4.1, Hoffmann et al. [93] 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 PRISM to verify properties about a pilotless aircraft on a foraging mission, with probabilistic modelling used to capture the physical environment. Markov chains have also been used to build models for predicting the likelihood of a robot swarm aggregating [48]. Each robot is represented by a Markov chain, where the probabilities to join or leave an aggregate are a function of the size of the aggregate (larger aggregates are preferred). The authors of [48] find that their model correlates closely with results from simulations.

5.3 Logics

Logics are the second most prevalent formalism found during our literature search, the majority of which are some variety of temporal logic. We also found work using dynamic logics and a variety of logics that didn't warrant their own category (e.g. FOL). Temporal logics are discussed in §5.3.1, dynamic logics in §5.3.2, and other logics in 5.3.3.

5.3.1 Temporal Logics.

Temporal logics are used for specifying dynamic properties about a system over linear or branching time, which feature heavily in the literature (Table 2). There are a variety of temporal logics: Linear-time Temporal Logic (LTL) and Computation Tree Logic (CTL) deal with events occurring next, globally, or eventually; extensions like Probabilistic Temporal Logic (PTL) and Probabilistic Computation Tree Logic (PCTL) add notions of probability. Temporal logics have been used extensively to address the challenges that we identified in §3 and §4.

Modelling a robot's environment is a key challenge when building robotic systems (§3.1). One approach [83] uses a BA to model the environment and synthesises a motion controller by model-checking it to find an accepting path that satisfies an LTL task specification. The BA is updated at runtime with new environmental information and checked again to revise the motion controller.

The work in [99] presents a probabilistic approach to modelling a BDI robotic agent and its environment. The agent can be captured as either a Discrete-Time Markov Chain (DTMC) or a Markov Decision Process (MDP). Then, they use PCTL to specify properties, which are model-checked, using PRISM. They apply this technique to an autonomous mine detector, showing how their approach can be used for both design-time checking and runtime monitoring (which we discuss in §6.3). The runtime safety monitors for autonomous systems presented in [122] are specified in CTL. The work in [58], aimed generally at safe robotics, captures assumptions about the real world using Signal Temporal Logic (STL) and uses them for runtime monitoring. Another example, this time aimed at robot swarms [120], models the system's potential behaviour and constrains it according to Supervisory Control Theory (SCT) specifications of safe behaviour.

Several approaches use temporal logic to provide certification evidence or build public trust in autonomous robotic systems. The work in [173, 176] captures air safety rules and assumptions using LTL (extended to capture BDI agent beliefs) which are used to verify that an autonomous pilotless aircraft follows the rules of the air in the same way as a pilot. Similarly, [60] and [76] present an approach for automatically building PTL models of the safety rules and environment of a robotic domestic assistant. These models are checked against a probabilistic description of the robot's environment (a sensor-equipped house in the UK) to ensure that the rules are sufficient to keep a human safe. An extension [175] checks models of the rules against environment models based on data collected from a person living in the house for several days. These verification results improve the confidence in the safety of the robot's high-level decision making.

Enabling robots to deal with ethical choices is a current open challenge. One approach presents a BDI language, ETHAN [55], for reasoning about ethical principles. It can be verified, using AJPF, that when an ETHAN agent chooses a plan, all other such plans are ethically worse.

Another popular approach is synthesising behavioural models that satisfy temporal logic properties. For example, in [104] motion plans are built incrementally that guide the robot to its goal position, while satisfying properties that are specified in a deterministic subset of μ-calculus. There is potential for using this approach at runtime; a plan can be built from a system of over 1000 states in ~3.5 seconds. In [116], LTL specifications are used to synthesise robot motion automata. This approach mediates state explosion by planning a short distance ahead, repeating after each plan is complete. The work in [141] presents a tool that generates hybrid robot controllers from LTL specifications, which can be used in simulation or with real robots.

Temporal logics have also been used to verify particular robotic software frameworks. The existing tool-chain for G^{en}oM has been extended to generate Fiacre models of a system, which can then be model-checked against LTL properties [75]. Another approach models ROS nodes, and the communication between them, as TA; which are checkd against TCTL properties in UPPAAL [86]. They verify the lack of message buffer overflows for a selection of queue sizes in the open-source robotic application Kobuki.

Temporal logics have been used to help overcome some of the challenges in developing swarm robotic systems. Winfield et al. describe a *dependable swarm* [184]: a distributed multi-robot system, based on the principles of swarm intelligence, that we can rely on the behaviour of. They advocate the use of temporal logic 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 its specification. This is a common approach taken when using formal methods, since modelling continuous elements of the system is difficult. In a companion paper [62] they apply model-checking techniques to the alpha swarm navigation algorithm.

As mentioned in §4.2.1, swarm robotic systems exhibit both macroscopic and microscopic properties. The authors of [128] 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 the swarm's environment as a *region graph*, describes the swarm's movement using *region propositions*, and specifies the swarm's macroscopic and microscopic behaviour (separately) using LTL. These components are combined to produce decentralised controllers for the robot swarm.

In contrast, [70] presents a technique for dealing with decentralised control in multi-agent systems that keeps the specifications local to each robotic agent. Each local specification is written in LTL, with their mutual satisfiability not guaranteed beforehand. The local specifications are exchanged by the robots at physical meeting points and model-checked for mutual satisfiability. However, this technique assumes that dependencies between robotic agents are sparse, so it may not scale well in robot swarms or teams with a larger number of dependencies.

Probabilistic temporal logics have proved useful in modelling robot swarms [118]. The work in [110] models the probabilistic state machine from [118], which is checked in PRISM for satisfaction of PCTL properties. Since each robot in the swarm is characterised by the same state machine, a *counting abstraction* is used to reduce the state space; but, this abstraction only works for homogeneous swarms.

The work in [35] presents a top-down swarm development methodology that begins with a PCTL specification of the swarm's macroscopic requirements. Successively, more concrete models of the swarm are developed, including simulation and field test implementations. Each step is checked against the previous steps to ensure that 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.

5.3.2 Dynamic Logic.

Dynamic logic is an extension of modal logic that can be used to specify and reason about properties of dynamic systems thus they are useful for specifying robotic systems.

Mitsch et al. use dL [135], which was designed for the specification and verification of hybrid programs, to describe the discrete and continuous navigation behaviour of a ground robot [125, 126]. Mitsch et al. use the hybrid theorem prover KeYmaera [125] (and its sucessor, KeYmaera X [126]) to prove safety (and liveness) properties about a robot's behaviour, written in dL. They also describe an automatic synthesis, from the verified dL model, to runtime monitors for safety properties [126].

Other work uses quantified differential dynamic logic QdL to analyse a control algorithm of a surgical robot [113]. They also used a variation of KeYmaera, called KeYmaeraD, for proof support.

5.3.3 Other Logics.

We identified a number of other logics that were not numerous enough to for their own category.

Dylla et al. [64] formalise the basic concepts of football theory (from a leading football theory manual) using Readylog – which is a variant of the logic programming language, Golog. They use this formalisation to formally describe football tactics, which they implement on football-playing robots for the RoboCup competition.

Gjondrekaj et al. [80] use a formal language that has been designed to capture properties about distributed systems, KLAIM, its stochastic extension, STOKLAIM, and their related tool set. Their application domain is distributed robotic systems and they describe the modelling of three homogeneous robots, under distributed control, cooperating to transport objects.

Talamadupula et al. [164] use FOL to capture an agent's beliefs and intentions. They use this to enable the prediction of other agents' plans.

5.4 Process Algebras

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. Process algebras can capture (discrete) time, but they seldom 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) [7]. FSP specifies the required safety and liveness properties, which are transformed into Labelled-Transition Systems (LTSs); then the agent program and architecture (writted in πADL) are checked to see if they satisfy these properties.

The Bio-PEPA process algebra has been used to model the microscopic behaviour of a robot swarm [123]. It enables several different analysis methods using the same specification. They describe a specification of a robot swarm performing a foraging task where a team of three robots is required to collect each object, 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. This collection of results is compared with an 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 [91], for a timed state machine notation [144]. This is a similar approach to the (non-formal) state chart notations ArmarX [171] and rFSM [124], described in §2. RoboChart is supported by an Eclipse-based environment, RoboTool [127], which allows the graphical construction of RoboChart diagrams that it can be, automatically, translated 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 [143].

The CSP model-checker, the Failures-Divergences Refinement checker (FDR) [79], can be opened from within RoboTool to enable quick checking of the timed and untimed properties of the RoboChart model. RoboTool automatically generates basic assertions (to check for deadlock, livelock, timelock, and non-determinism) for FDR. These can be edited, but this requires some skill with CSP and knowledge of the model, which many software engineers do 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 [6] derives correct and complete CSP+T specifications from a description of a system in the Unified Modelling Language (UML) profile UML-RT – designed for developing real-time systems. This work maps each subsystem in the UML-RT description of the system to a CSP+T process, and composes them in parallel. They

demonstrate this approach on a two-armed industrial robot. However this mapping appears to be a manual, informal process.

As previously mentioned in §5.2, the graphical robot mission design tool MissionLab contains a process algebra, PARS. The work in [121] 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.

5.5 Ontologies

Ontologies act as a body of knowledge and provide a vocabulary for a given domain [85]. They formally specify the "key concepts, properties, relationships and axioms of a given domain" and enable reasoning over this knowledge to infer new information [137].

For autonomous robotics, ontologies have been used to describe the robot environment, describe and reason about actions, and for the reuse of domain knowledge [137]. Ontologies also have the benefit of being able to capture elements of Human-Robot Interaction. Despite finding ontologies for autonomous robotics systems, our literature search found little evidence of available tool support.

The IEEE-RAS working group *Ontologies for Robotics and Automation* has published an ontology[4] to explicitly and formally specify the shared concepts within robotics and automation [85, 137].

KNOWROB is a knowledge processing system for autonomous personal robotic assistants [166] where knowledge is represented in description logic using Web Ontology Language (OWL). Their system can be used with ROS and integrates encyclopaedic knowledge, an environment model, action-based reasoning and human observations. In [11], they use ontological knowledge, represented using the OWL, of a manufacturing environment to enable reconfiguration without human intervention. This work uses OWL reasoners in order to decide how best to reconfigure the system.

5.6 Other Formalisms

Besides those described in the previous subsections, a number of other formalisms and approaches feature in the literature. Some of these formalisms are used to specify the system and others are used to specify the properties the system is being checked for. In the literature we found, each of the studies that used 'other' formalisms only specify either the system or the properties, not both.

As previously mentioned in § 4.1, autonomous robotic systems are often programmed using agents. Several works we found [55, 57, 71, 176] make use of the AJPF [56] program model checker. The properties being checked by AJPF are described in temporal logic, but the system is a BDI agent program written in GWENDOLEN. The AJPF translates agent programs into a formally-defined internal representation to enable model checking.

The work in [138] describes the verification of safety properties using the Averest framework. Robotic systems are described using the Quartz language and the safety properties are specified in μ-calculus. Quartz[5] is a synchronous language that allows the developer to describe, for example, parallel execution and nondeterministic choice.

Several papers use the Behaviour Interaction Priority (BIP) framework [19], which uses familiar FSMs to specify a system but provides the D-Finder tool [26] to check for deadlocks and other safety properties. Some work shows how BIP can be used to specify and verify robotic software [18, 20, 25]. In addition, RV-BIP [66] is a tool that produces a runtime monitor as an "additional component in a BIP system" that can check an LTS-based specification at runtime. Finally, the work in [1] provides a synthesis technique to generate robotic software from BIP models.

---

[4]IEEE 1872-2015 - IEEE Standard Ontologies for Robotics and Automation https://standards.ieee.org/standard/1872-2015.html
[5]http://www.averest.org/#about_thesynchronouslanguagequartz

Another synthesis technique is presented in [40] and [41], which tackle robotic teams' communication strategies. The properties that the robotic team must exhibit (that is, the tasks) are specified using a (formally defined) regular expression language. These specifications are used to synthesise robot controllers, as FSA.

RELAX is a requirements language for dynamic self-adaptive systems that enables developers to indicate requirements that may be relaxed at run-time [180]. The syntax of RELAX is structured natural language and boolean expressions (including standard temporal operators). It's semantics is defined in terms of temporal fuzzy logic.

5.7 Summary of Formalisms for Robotic Systems

This section describes the formalisms currently used in the literature for specifying the behaviour of robotic systems. Table 2 summarises the formalisms found, according to the methodology described in §1.1, in use to specify the system and properties to be checked. There were more studies found that specify the system (56) than specify the properties (41). Some systems specifications are checked for deadlock or reachability, for example [49]; whereas some are just used for specification, such as [177]. Conversely, some of the studies only specify properties and monitor a running system [95, 117], or describe a framework for dealing with property specification [180]. Also, the ontologies found in our literature search are all used to specify systems.

State-transition systems are the most numerous for specifying systems and temporal logics the most numerous for specifying properties. This popularity may be explained by the popularity and usability of model-checkers for these formalisms; state-transition systems are often used to describe a system's behaviour, each state of which is then checked to see if a temporal logic property holds. This is shown in Table 3, where model-checkers are the most used formal verification tool.

Despite state-transition systems and temporal logics dominating the literature found, Table 3 shows that there are still a wide variety of tools being used for these formalisms. This may be because different model-checkers capture different features, such as time or probability. But this possibly presents a challenge to the sharing and extension of formal models.

6 FORMAL VERIFICATION APPROACHES FOR ROBOTICS

This section analyses the surveyed literature in terms of the verification approaches that were used. Here, "approach" refers to the framework(s), technique(s) or combination thereof that were used to verify that the model or implementation of the system being developed preserves the required properties; where §5 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 §5 are different kinds of bricks, then the approaches that we describe here are the types of buildings that can be constructed from those bricks.

We explore several distinct approaches to formal verification of robotic systems as illustrated in Table 4. Specifically, §6.1 describes model-checking approaches, §6.2 describes approaches using theorem proving, §6.3 discusses approaches using runtime verification monitors, §6.4 discusses approaches that use a combination of different formalisms (e.g. CSP and B) or approaches (e.g. model-checking and theorem proving), §6.5 outlines frameworks for building verifiable robotic software; finally, §6.6 summarises and answers the remaining research questions (RQ2 and RQ3).

6.1 Model-Checking

Model-checking is the most widely used approach to verifying robotic systems; it has been used with temporal logics [60, 99], process algebras [127], and programs [68, 71]. Arguably, the popularity of model-checking owes something to the volume of publications that we found using temporal logic (Table 2), which are often used in model-checking approaches. Further, we see two main reasons

Approach	References	Total
Model-Checking	[110], [123], [111], [36], [61], [62], [97], [98] [51], [89], [174], [175], [50], [70], [176], [138], [139], [43], [24], [18], [25], [1], [40], [30], [104], [71], [55], [57], [176], [114], [44], [129]	32
Theorem Proving	[125], [89], [113]	3
Runtime Monitoring	[95], [66], [117]	3
Integrated Formal Methods	[51],[43], [6],[29], [89], [81], [97], [117]	8
Formal Software Frameworks /Architectures	[66], [141], [24], [179], [20], [18], [25], [138], [139], [178]	10

Table 4. Summary of the approaches to formal verification found using the methodology described in §1.1. Note: some of the surveyed papers use more than one approach.

for *actively* choosing model-checking. First, model-checkers are automatic, 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.

Some model-checkers can handle timed models (e.g. Uppaal) or probabilistic models (e.g. PRISM). 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 [170].

Model-checking exhaustively explores the state space, so one must be careful about the input models and properties to be checked. State explosion can be crippling to verification efforts using model-checking. Webster et al. [174] report requiring 10^4 MB of memory to verify each individual safety property of their domestic assistant robot model. Similar state explosion problems are present in models of swarm robotic systems [110]. Various standard approaches can mitigate state space explosion and keep models tractable [45, 46]. For example, for swarms, symmetry reduction [15] or abstracting the swarm to a single state machine with a counting abstraction can help [173].

Model-checking has been used to build behaviour specifications (e.g. robot movement plans). One approach uses model-checking to build robot motion plans that satisfy properties specified in a deterministic subset of μ-calculus [104]. 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. Similarly, in [83], a BA representing the robot's environment is model-checked for an accepting path satisfying an LTL specification. The accepting path is used to synthesise a motion controller, which is then revised at runtime by repeated model-checking and synthesis. Kloetzer et al. [108] use model-checking to find traces of a transition system describing the behaviour of a robot team that satisfy an LTL-X formula. This trace is 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 offers 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 [147]. More recently, the RoboChart notation and it's associated toolset also makes use of Isabelle/HOL to verify robotic systems [74].

Other work in this domain includes [125], which uses the hybrid systems logic dL [135] to describe the discrete and continuous navigation behaviour of a robot rover. They then use the hybrid theorem prover, KeYmaera, to verify that the robot does not collide with stationary or moving obstacles, and maintains sufficient distance from obstacles. Mitsch et al. [126] 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.

In [89], OZS (a combination of Object-Z and Statecharts [87]) is used to capture the dynamic roles that an agent in a multi-robot system can play, which define the robot's interaction patterns. OZS has an operational semantics, and the STeP theorem prover is used for verification of safety and liveness properties of their model.

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 complement 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 ensuring safe operation [158].

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 runtime verification, which brings together model-checking and stream-processing technologies [148]. Examples include [66, 95, 97], and the International Conference on Runtime Verification has been running since 2006.

Aniculaesei et al. [14] 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. [69] 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 (with some environmental assumptions) becomes invalid.

Kane et al. [103] present runtime monitors for an autonomous research vehicle. These monitors are written in the $\alpha \mathcal{VSL}$ safety specification language, whose semantics is given over time-stamped traces using future-bounded, propositional Metric Temporal Logic (MTL). Their algorithm, EgMon,

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 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 [21] describe a method of distributing an LTL specification of a system's macro-level behaviour to a collection of micro-level behavioural monitors. Their approach does not assume any central information collection and ensures 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 [158], which describes an approach for modelling 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 illustrate their approach using an electronically controlled train braking system. A similar hybrid-logic approach is taken by [126], who specify the safety properties of a robot rover using dL and then automatically synthesise runtime monitors from these verified models.

ROSRV is a runtime verification framework for robotics systems deployed on ROS [95] that uses a novel formal specification language for writing safety properties. Using a new 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 made 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.

Similarly, Falcone et al. [66] describe RV-BIP, a tool that produces runtime monitors for robots in the BIP framework, which will be discussed in §6.5. 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, *currently* true or false. RV-BIP takes 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. [117] 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 semi- or non-formal approach, that complement each other. While still difficult, iFM can capture several dimensions of a system at once (for example static 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 [67]. 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 [90].

The challenges presented in §4 are often best tackled using iFM. However, our literature search found no generic approaches for applying iFM to robotics. Here, we discuss some notable bespoke examples. To enable both refinement-based development and probabilistic checking, [165] uses

both Event-B and a probabilistic language to check reconfigurable architectures for an on-board satellite system. The work in [7] 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 [159] 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 [101], an agent controlling a car is verified using AJPF, and the timing properties are verified using UPPAAL. This is extended, in [102] 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) [47].

Similarly, OZS combines Object-Z [89] and Statecharts [87]; it has been used to describe the state and behaviour of systems is used to formalise part of the Satisfaction-Altruism Model [157]. This model captures the dynamic set of roles that a robot in a multi-robot team can perform. The roles define the interaction pattern of the robot. OZS has an operational semantics, which enables theorem proving. In [89], the authors describe verification of their model using the SAL model-checker and the STeP theorem prover. They also indicate an intent to provide a way to refine OZS models to code for deployment on robots.

These bespoke examples integrate two or more formal methods to obtain an approach to verification 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 [6], 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.

RoboChart is a robotic system design notation that integrates CSP with a graphical timed state machine notation [144]. 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 reusable, stable and generic approach to iFM is key to the success iFMs, not just for robotic systems but more generally as well. Integrating diverse formalisms into a holistic framework remains an open problem, waiting to be solved.

In the area of automated surgical robotics the combination of UML, LTL, and Alloy has been used to model and reason about a number of surgical operations [29]. The work employs a goal-oriented methodology where a UML model relates goal requirements to the "high-level structural and behavioural decomposition of the intelligent robotic system". A number of experts (medical professionals) were interviewed and their responses were used to construct a goal model which was converted into a set of formal properties using the Alloy language where each goal is formally defined in LTL. This model is then checked by the KodKod SAT solver. The UML model can then be translated into executable code using the Orocos framework.

The Restore Invariant Approach (RIA) uses invariants to specify productive states and to use constraint solving techniques whenever the system is not operational to find new configurations to make the system operational again [81]. This work targets self-adaptive and agent-based systems. It was used in the specification and verification of an adaptive production cell that was made up of three drill robots and two autonomous transportation units that connect them. Here, the authors make use of class diagrams with Object Constraint Language (OCL) constraints, the Microsoft Robotics Studio simulator, JADEX multi-agent framework and the Alloy constraint solver.

6.5 Frameworks for Verifiable Robotic Software

Our literature search revealed a number of frameworks for developing verifiable robotic systems. These frameworks often encompass a number of the techniques already described in §6, but

frequently, they incorporate bespoke tools and formalisms. Although some facilitate the use of multiple verification techniques, we omit them from §6.4 since users are generally expected to apply only one of the available techniques in practice rather than integrating results from several.

The BIP framework [19] is a toolset for modelling component-based real-time software, with a notation based on FSMs. The basic component of a BIP model 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 D-Finder tool [26]. 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 constraints in the BIP specification are enforced by the generated software.

Other work along this vein includes a metamodel-based translation from the Architecture Analysis and Design Language (AADL) into BIP [43]. 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. [66] integrate runtime verification into the BIP framework. They describe their approach and a tool, RV-BIP, which generates monitors for BIP to check specifications at runtime. Related to this are attempts to integrate BIP and the LAAS architecture which is based on G^{en}oM [20, 25].

The Averest framework provides "tools for verifying temporal properties of synchronous programs" that are written in the Quartz language [139]. The 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. [40, 41] propose a computational framework for automatic synthesis of control and communication strategies for robot teams, using task specifications that are written as regular expressions. Their objective is to "generate provably correct individual control and communication strategies" from regular expressions, which are used to create FSA. They demonstrate their approach using a Robotic Urban-Like Environment (RULE), where autonomous cars must navigate and avoid collisions. Their technique builds on LTL model-checking and uses JFLAP and MatLab [41].

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 [179]. This is built upon MAPE-K and supports agents and formal refinement of specifications in Z.

Kim et al. [106] use 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.

6.6 Summary of Approaches

This section discussed the formal verification approaches in the literature as summarised in Table 4. The most popular approach is model-checking, which is reflected in the dominance of temporal logics and state-transition systems for the specification of robotic systems (§5, Table 2). We speculate that the prevalence of model-checking in the literature may be because it is easy for developers to understand and trust that do not have a formal methods background since it is automatic and conceptually similar to exhaustive testing. Frameworks for verifiable robotic systems were the next most popular approach and most of these incorporated a bespoke model-checker. However, it is not clear, in practice, just how effective these inbuilt verification tools are.

As outlined earlier, iFM are necessary in the verification of robotic systems due to their size and complexity. This is an active area of research with tools still under development. We anticipate that, in the future, this approach will feature more in the verification of autonomous robotic systems. The least popular approaches were theorem proving and runtime monitoring. In general, theorem provers require specialist knowledge to use and we believe that this is why it is currently not as popular as other approaches. Although effective in the running system, runtime monitoring is often seen as a separate step in the verification process after static verification has taken place and provides little in the way of design-time benefits.

7 CONCLUSION

§1.2 discussed related surveys and differentiated them from our work. Our work incorporates the topics of formal methods for self-adaptive systems (surveyed in [177]) and swarms (from [149, 150]). We also identify work from several categories that a survey on safety-critical robotics [82] lists as areas to focus on. Specifically, these are modelling of a robot's physical environment, formal verification of robotic systems, correct-by-construction controller synthesis, and being able to provide certification evidence. This last point is particularly important, and is a key focus in the current literature. Finally, the work that we have surveyed is squarely within the category of verifying applications for internal correctness; one of three main challenges for the automatic verification of autonomous systems [156]. Some of the current literature focusses on the external correctness of the software, for example how it will safely interact with a robotic software framework like ROS or $G^{en}oM$. However, the suggestions that special purpose language interpreters or compilers must be verified has not been borne out, as many robotic applications are written in general purpose languages such as Python, Java, or C++. This, of course, leaves the tools for these languages to be verified, a larger challenge due to the generality of the languages.

§1.1 outlined the research questions that this paper investigates. In this section, we use the results of our survey to derive potential answers to these research questions. The answers also provide the structure for a discussion of our observations of the current state of the literature.

RQ1, asked what the challenges are when formally specifying and verifying (autonomous) robotic systems. We identified two types of challenge: those *external* to the robotic system, discussed in §3; and those *internal* to the robotic system, discussed in §4. The external challenges that we identified are: modelling the robotic system's external environment, and providing sufficient evidence for public trust and certification. The internal challenges that we identified are the use of: agent-based, multi-robot, and adaptive/reconfigurable systems. Reconfigurability is key to safely deploying robots in hazardous environments and vastly more work needs to materialise in order to ensure the safety of reconfigurable autonomous systems. Therefore, we see a clear link between a robotic system reacting to the changes in its external environment, and reconfigurable systems.

Similarly, *rational* agent-based systems that can explain their reasoning provide a good route for providing evidence for public trust or certification bodies. This is because they enable the crucial transparency that trust and certification needs. An example of the transparency required is the *ethical black box* proposed by [181], which records the sensor input and internal state of the system. A rational agent can provide reasons for its choices, based in the input and internal state information.

RQ2, asked what are the current formal methods used for tackling the challenges identified by answering **RQ1**. Our analysis, presented in §5 and §6, shows that temporal logics, state-transition systems, and model-checkers are the most prominent formalisms and approach in the literature over the past decade. We speculate that this is due to the fact that temporal logics and state-transition

systems allow abstract specification, which is useful earlier in the development process, and because model-checking as an approach is generally easy to explain to stakeholders who do not have experience using formal methods.

RQ3, asked what the limitations are of the best practice formalisms and approaches to verification that were identified in the answer to **RQ2**. One limitation appears to be a resistance to adopting formal methods in robotic systems development [120]. The perception is that applying formal methods is a complicated additional step in the engineering process, 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. These are long-standing opinions, for example the authors of [109], which was published 19 years prior to [120]. There are, however, notable examples of industrial uses of formal methods, such as those examined in [185].

The wide variety of tools for the same formalism (indicated in Table 3) also points to a problem: the lack of interoperability between formalisms. Often, models or specifications of similar components 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 in smoothing the conversion between formalisms or tools. Further, this would serve a growing need to capture the behaviour of complex systems using a heterogeneous set of formalisms, each suited to the component being modelled or the properties of interest. This is currently an open problem in formal methods for robotic systems.

As mentioned in §3, there is a lack of clear guidance when it comes to choosing a suitable formal method for a particular system. With respect to autonomus robotic systems, Table 1 lists the formalisms/tools used and the corresponding case study that was presented in the surveyed literature. This provides some guidance when choosing an appropriate formal method for a given system but we leave a more detailed analysis of this as future work.

Another limitation faced by formal methods for robotic systems (and more generally) is that of formalising the *last link*, the step between a formal model and program code. We found few examples of formal methods for robotics that also produced runnable program code from a specification, and, in these examples, it was often unclear as to how the code relates back to the original specification model. Such last link translation steps require a formal underpinning to ensure that the model is represented by the code. The lack of clarity about this limitation points to another: a lack of open sharing of models, code, and realistic case studies that are not tuned for a particular formalism.

Field tests and experiments using simulations are both useful tools for robotic systems development; but formal verification is crucial, especially at the early stages of development when field tests of the control software are infeasible (or dangerous). A focussed research effort on the combination or integration of formal methods should improve their use in robotic systems development, because no one formalism is capable of adequately capturing that all aspects of a robotic system behave as expected. Ensuring that these tools are usable by developers and providing similar features in an IDE would also improve their uptake by simplifying their use. Work in this area could lead to an Integrated *Verification* Environment, allowing the use of different formalisms using same developer front-end, connecting them to their respective tools, and providing helpful IDE-like support.

We have identified a number of *threats to validity* for this work, and we have taken measures to mitigate them where possible. In particular, researcher bias was minimised by explicitly stating the scope and research questions at the beginning of the review and having two researchers analyse the search results in tandem. Restricting our search to those papers published within the last ten years may affect the completeness of our search results, however, by 'snowballing' we have explored and discussed papers that did not fall within this range throughout the narrative in this report.

Furthermore, our use of Google Scholar limits our search to those papers that were found using Google's algorithms. Again, our use of "snowballing" helps to mitigate this threat.

This paper has analysed the current state-of-the art literature particular to the formal specification and verification of autonomous robotic systems. To this end, this survey identifies future directions in this field.

REFERENCES

[1] T. Abdellatif, S. Bensalem, J. Combaz, L. De Silva, and F. Ingrand. Rigorous design of robot software: A formal component-based approach. *Rob. Auton. Syst.*, 60(12):1563–1578, 2012.

[2] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 1st edition, 2010.

[3] K. Adam, K. Olldobler, B. Rumpe, and A. Wortmann. Modeling Robotics Software Architectures with Modular Model Transformations. *J. Softw. Eng. Robot.*, 8(1):3–16, 2017.

[4] S. Adam, M. Larsen, K. Jensen, and U. P. Schultz. Rule-based Dynamic Safety Monitoring for Mobile Robots. *J. Softw. Eng. Robot.*, 7(1):120–141, 2016.

[5] H. S. Ahn, M. H. Lee, and B. MacDonald. Design of a Robotic Software Manager for using Heterogeneous Components of Different Frameworks. *J. Softw. Eng. Robot.*, 8(1):45–64, 2017.

[6] K. B. Akhlaki, M. I. C. Tuñón, J. A. H. Terriza, and L. E. M. Morales. A methodological approach to the formal specification of real-time systems by transformation of UML-RT design models. *Sci. Comput. Program.*, 65(1):41–56, 2007.

[7] N. Akhtar. Contribution to the Formal Specification and Verification of a Multi-Agent Robotic System. *Eur. J. Sci. Res.*, 117(1):35–55, 2014.

[8] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *J. Rob. Res.*, 17(4):315–337, 1998.

[9] R. Alami, K. M. Krishna, and T. Siméon. Provably Safe Motions Strategies for Mobile Robots in Dynamic Domains. In *Auton. Navig. Dyn. Environ.*, volume 35 of *STAR*, pages 85–106. Springer, 2007.

[10] S. Alexandrova, Z. Tatlock, and M. Cakmak. RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In *Robot. Autom.*, pages 5537–5544. IEEE, 2015.

[11] Y. Alsafi and V. Vyatkin. Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing. *Robot. Comput. Integr. Manuf.*, 26(4):381–391, 2010.

[12] M. Althoff, D. Althoff, D. Wollherr, and M. Buss. Safety verification of autonomous vehicles for coordinated evasive maneuvers. In *Intell. Veh. Symp.*, pages 1078–1083. IEEE, 2010.

[13] N. Ando, T. Suehiro, and T. Kotoku. A software platform for component based RT-system development: OpenRTM-Aist. volume 5325 of *LNCS*, pages 87–98. Springer, 2008.

[14] A. Aniculaesei, D. Arnsberger, F. Howar, and A. Rausch. Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments. *Theor. Comput. Sci.*, 232:79–90, 2016.

[15] L. Antuña, D. Araiza-Illan, S. Campos, and K. Eder. Symmetry reduction enables model checking of more complex emergent behaviours of swarm navigation algorithms. volume 9287 of *LNCS*, pages 26–37. Springer, 2015.

[16] R. Arkin, P. Ulam, and B. Duncan. An Ethical Governor for Constraining Lethal Action in an Autonomous System. Technical Report GIT-GVU-09-02, Georgia Inst. of Tech., 2009.

[17] R. Arkin, P. Ulam, and A. Wagner. Moral Decision Making in Autonomous Systems: Enforcement, Moral Emotions, Dignity, Trust, and Deception. *Proc. IEEE*, 100(3):571–589, 2012.

[18] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-h. Nguyen, and J. Sifakis. Rigorous System Design Using the BIP Framework. *Software*, 28(3):41–48, 2011.

[19] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Softw. Eng. Form. Methods*, pages 3–12. IEEE, 2006.

[20] A. Basu, M. Gallien, C. Lesire, T.-H. Nguyen, S. Bensalem, F. Ingrand, and J. Sifakis. Incremental Component-Based Construction and Verification of a Robotic System. In *ECAI*, volume 178, pages 631–635, 2008.

[21] A. Bauer and Y. Falcone. Decentralised LTL Monitoring. *Form. Methods Syst. Des.*, 48(1-2):46–93, 2016.

[22] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas. Symbolic planning and control of robot motion [Grand Challenges of Robotics]. *Robot. Autom. Mag.*, 14(1):61–70, 2007.

[23] G. Beni. From Swarm Intelligence to Swarm Robotics. In *Swarm Robot.*, volume 3342 of *LNCS*, pages 1–9. Springer, 2005.

[24] S. Bensalem, L. De Silva, A. Griesmayer, F. Ingrand, A. Legay, and R. Yan. A formal approach for incremental construction with an application to autonomous robotic systems. In *Softw. Compos.*, volume 6708 of *LNCS*, pages 116–132. Springer, 2011.

[25] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, and N. Thanh-Hung. Toward a More Dependable Software Architecture for Autonomous Robots. *Robot. Autom. Mag.*, 16(1):67–77, 2009.

[26] S. Bensalem, A. Griesmayer, A. Legay, T. H. Nguyen, J. Sifakis, and R. Yan. D-finder 2: Towards efficient correctness of incremental design. In *NASA Formal Methods Symposium*, volume 6617 LNCS, pages 453–458, 2011.

[27] Z. Bi, S. Lang, M. Verner, and P. Orban. Development of reconfigurable machines. *Adv. Manuf. Technol.*, 39(11-12):1227–1251, 2008.

[28] G. Biggs and B. MacDonald. Specifying robot reactivity in procedural languages. In *Intell. Robot. Syst.*, pages 3735–3740. IEEE, 2006.

[29] M. Bonfe, F. Boriero, R. Dodi, P. Fiorini, A. Morandi, R. Muradore, L. Pasquale, A. Sanna, and C. Secchi. Towards automated surgical robotics: A requirements engineering approach. In *Biomed. Robot. Biomechatronics*, pages 56–61. IEEE, 2012.

[30] R. Bordini, M. Fisher, and M. Sierhuis. Formal verification of human-robot teamwork. In *Hum. Robot Interact.*, pages 267–268. ACM, 2009.

[31] S. Bouraine, T. Fraichard, and H. Salhi. Provably safe navigation for mobile robots with limited field-of-views in dynamic environments. *Auton. Robots*, 32(3):267–283, 2012.

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

[33] R. Brafman, M. Bar-Sinai, and M. Ashkenazi. Performance level profiles: A formal language for describing the expected performance of functional modules. In *Intell. Robot. Syst.*, pages 1751–1756. IEEE, 2016.

[34] J. Braman, R. Murray, and D. Wagner. Safety verification of a fault tolerant reconfigurable autonomous goal-based robotic control system. In *Intell. Robot. Syst.*, pages 853–858. IEEE, 2007.

[35] M. Brambilla, A. Brutschy, M. Dorigo, and M. Birattari. Property-Driven Design for Robot Swarms. *Trans. Auton. Adapt. Syst.*, 9(4):1–28, 2014.

[36] M. Brambilla, C. Pinciroli, M. Birattari, and M. Dorigo. Property-driven design for swarm robotics. In *Auton. Agents Multiagent Syst.*, 2012.

[37] D. Brutzman, D. Davis, G. Lucas, and R. McGhee. Run-time Ethics Checking for Autonomous Unmanned Vehicles: Developing a Practical Approach. In *Unmanned Untethered Submers. Technol.* Monterey, California: Naval Postgraduate School., 2013.

[38] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali. The BRICS component model. In *Symp. Appl. Comput.*, page 1758. ACM, 2013.

[39] J. Celaya, A. Desrochers, and R. Graves. Modeling and analysis of multi-agent systems using petri nets. In *Conf. Syst. Man Cybern.*, volume 4, pages 1439–1444. IEEE, 2007.

[40] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *Trans. Robot.*, 28(1):158–171, 2012.

[41] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. A Formal Approach to Deployment of Robotic Teams in an Urban-Like Environment. In *Distrib. Auton. Robot. Syst.*, volume 83 of *STAR*, pages 313–327. Springer, 2013.

[42] B. Cheng, K. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. Müller, P. Pelliccione, A. Perini, N. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. Villegas. Using Models at Runtime to Address Assurance for Self-Adaptive Systems. In *Models@run.time*, volume 8378 of *LNCS*, pages 101–136. Springer, 2014.

[43] M. Y. Chkouri, A. Robert, M. Bozga, and J. Sifakis. Translating AADL into BIP-application to the verification of real-time systems. In *Model Driven Eng. Lang. Syst.*, volume 5421 of *LNCS*, pages 5–19. Springer, 2008.

[44] J. Choi, S. Kim, and A. Tsourdos. Verification of heterogeneous multi-agent system using MCMAS. *J. Syst. Sci.*, 46(4):634–651, 2015.

[45] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *J. ACM*, 50(5):752–794, 2003.

[46] E. Clarke, O. Grumberg, and D. Long. Model Checking and Abstraction. *Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[47] S. Colin, A. Lanoix, O. Kouchnarenko, and J. Souquières. Using CSP∥B Components: Application to a Platoon of Vehicles. In *Form. Methods Ind. Crit. Syst.*, volume 5596 of *LNCS*, pages 103–118. Springer, 2009.

[48] N. Correll and A. Martinoli. Modeling and designing self-organized aggregation in a swarm of miniature robots. In *Robot. Autom.*, volume 30, page 6, 2007.

[49] H. Costelha and P. Lima. Modelling, analysis and execution of multi-robot tasks using petri nets. In *Auton. Agents Multiagent Syst.*, volume 41, pages 1187–1190. IEEE, 2008.

[50] N. Dantam and M. Stilman. Robust and efficient communication for real-time multi-process robot software. In *Humanoid Robot.*, pages 316–322. IEEE, 2012.

[51] D. G. De La Iglesia and D. Weyns. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *Trans. Auton. Adapt. Syst.*, 10(3):15, 2015.

[52] E. Denney, G. Pai, and J. Pohl. Automating the Generation of Heterogeneous Aviation Safety Cases. *IEEE Trans. Reliab.*, 63(NASA/CR-2011-215983):1–20, 2011.

[53] E. Denney and S. Trac. A software safety certification tool for automatically generated guidance, navigation and control code. In *Aerosp. Conf.*, pages 1–11. IEEE, 2008.

[54] L. Dennis and B. Farwer. Gwendolen : A BDI Language for Verifiable Agents. *Log. Simul. Interact. Reason.*, pages 16–23, 2008.

[55] L. Dennis, M. Fisher, M. Slavkovik, and M. Webster. Formal verification of ethical choices in autonomous systems. *Rob. Auton. Syst.*, 77:1–14, 2016.

[56] L. Dennis, M. Fisher, M. Webster, and R. Bordini. Model checking agent programming languages. *Autom. Softw. Eng.*, 19(1):5–63, 2012.

[57] L. Dennis, M. Fisher, and A. Winfield. Towards Verifiably Ethical Robot Behaviour. *Artif. Intell. Ethics*, abs/1504.0(2014):1–11, 2015.

[58] A. Desai, T. Dreossi, and S. Seshia. Combining Model Checking and Runtime Verification for Safe Robotics. In *Runtime Verif.*, volume 10548 of *LNCS*, pages 172–189. Springer, 2017.

[59] M. D'Inverno, M. Luck, M. Georgeff, D. Kinny, and M. Wooldridge. The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System. *Auton. Agent. Multi. Agent. Syst.*, 9(1/2):5–53, 2004.

[60] C. Dixon, M. Webster, J. Saunders, M. Fisher, and K. Dautenhahn. "The fridge door is open" - Temporal verification of a robotic assistant's behaviours. volume 8717 of *LNAI*, pages 97–108. Springer, 2014.

[61] C. Dixon, A. Winfield, and M. Fisher. Towards temporal verification of emergent behaviours in swarm robotic systems. volume 6856 of *LNAI*, pages 336–347. Springer, 2011.

[62] C. Dixon, A. Winfield, M. Fisher, and C. Zeng. Towards temporal verification of swarm robotic systems. *Rob. Auton. Syst.*, 60(11):1429–1441, 2012.

[63] M. Dorigo and E. Şahin. Guest Editorial. *Auton. Robots*, 17(2-3):111–113, 2004.

[64] F. Dylla, A. Ferrein, G. Lakemeyer, J. Murray, O. Obst, T. Röfer, S. Schiffer, F. Stolzenburg, U. Visser, and T. Wagner. Approaching a formal soccer theory from behaviour specifications in robotic soccer. *Trans. State-of-the-art Sci. Eng.*, 32, 2008.

[65] 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 *Work. Softw. Eng. Adapt. Self-Managing Syst.*, pages 142–151, 2009.

[66] Y. Falcone, M. Jaber, T.-h. Nguyen, M. Bozga, and S. Bensalem. Runtime Verification of Component-Based Systems. In *Softw. Eng. Form. Methods*, volume 7041 of *LNCS*, pages 204–220. Springer, 2011.

[67] M. Farrell, M. Luckcuck, and M. Fisher. Robotics and Integrated Formal Methods: Necessity meets Opportunity. In C. Furia and K. Winter, editors, *Integr. Form. Methods*, volume 11023 of *LNCS*, page 10. Springer, 2018.

[68] L. Fernandes, V. Custodio, G. Alves, and M. Fisher. A Rational Agent Controlling an Autonomous Vehicle: Implementation and Formal Verification. *Theor. Comput. Sci.*, 257(Fvav):35–42, 2017.

[69] A. Ferrando, L. Dennis, D. Ancona, M. Fisher, and V. Mascardi. Recognising Assumption Violations in Autonomous Systems Verificaion. In *Auton. Agents Multiagent Syst.*, 2018.

[70] I. Filippidis, D. Dimarogonas, and K. Kyriakopoulos. Decentralized multi-agent control from local LTL specifications. In *Conf. Decis. Control*, pages 6235–6240. IEEE, 2012.

[71] M. Fisher, L. Dennis, and M. Webster. Verifying Autonomous Systems. *Commun. ACM*, 56(9):84–93, 2013.

[72] F. Fleurey, B. Baudry, R. France, and S. Ghosh. A Generic Approach for Automatic Model Composition. In *Model. Softw. Eng.*, volume 5002 of *LNCS*, pages 7–15. Springer, 2007.

[73] S. Fleury, M. Herrb, and R. Chatila. GenoM: A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture. In *Intell. Robot. Syst.*, volume 2, pages 842 – 849. IEEE, 1997.

[74] S. Foster, J. Baxter, A. Cavalcanti, A. Miyazawa, and J. Woodcock. Automating Verification of State Machines with Reactive Designs and Isabelle/UTP. *arXiv Prepr. arXiv1807.08588*, 2018.

[75] M. Foughali, B. Berthomieu, S. Dal Zilio, F. Ingrand, and A. Mallet. Model checking real-time properties on the functional layer of autonomous robots. In *Form. Methods Softw. Eng.*, volume 10009 of *LNCS*, pages 383–399. Springer, 2016.

[76] P. Gainer, C. Dixon, K. Dautenhahn, M. Fisher, U. Hustadt, J. Saunders, and M. Webster. CRutoN: Automatic Verification of a Robotic Assistant's Behaviours. In *Form. Methods Ind. Crit. Syst.*, volume 10471 of *LNCS*, pages 119–133. Springer, 2017.

[77] Y. Gao. *Contemporary planetary robotics: An approach toward autonomous systems*. 2016.

[78] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *Natl. Conf. Artif. Intell.*, volume 87, pages 677–682, 1987.

[79] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A. Roscoe. FDR3 - A Modern Model Checker for CSP. In *Tools Algorithms Constr. Anal. Syst.*, volume 8413 of *LNCS*, pages 187–201, 2014.

[80] E. Gjondrekaj, M. Loreti, R. Pugliese, F. Tiezzi, C. Pinciroli, M. Brambilla, M. Birattari, and M. Dorigo. Towards a formal verification methodology for collective robotic systems. In *Form. Eng. Methods*, volume 7635 of *LNCS*, pages 54–70. Springer, 2012.

[81] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif. A specification and construction paradigm for organic computing systems. In *Self-Adaptive Self-Organizing Syst.*, pages 233–242. IEEE, 2008.

[82] J. Guiochet, M. Machin, and H. Waeselynck. Safety-critical advanced robots: A survey. *Rob. Auton. Syst.*, 94:43–52, 2017.

[83] M. Guo, K. Johansson, and D. Dimarogonas. Revising Motion Planning under Linear Temporal Logic Specifications in Partially Known Workspaces. In *Robot. Autom.*, pages 5025–5032. IEEE, 2013.

[84] Y. Hafidi, L. Kahloul, M. Khalgui, Z. Li, K. Alnowibet, and T. Qu. On methodology for the verification of reconfigurable timed net condition/event systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, (99):1–15, 2018.

[85] T. Haidegger, M. Barreto, P. Gonçalves, M. Habib, S. K. V. Ragavan, H. Li, A. Vaccarella, R. Perrone, and E. Prestes. Applied ontologies and standards for service robots. *Rob. Auton. Syst.*, 61(11):1215–1223, 2013.

[86] R. Halder, J. Proença, N. MacEdo, and A. Santos. Formal Verification of ROS-Based Robotic Applications Using Timed-Automata. In *Form. Methods Softw. Eng.*, pages 44–50. IEEE, 2017.

[87] D. Harel. Statecharts: a visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[88] D. Heß, M. Althoff, and T. Sattel. Formal verification of maneuver automata for parameterized motion primitives. In *Intell. Robot. Syst.*, pages 1474–1481. IEEE, 2014.

[89] V. Hilaire, P. Gruer, A. Koukam, and O. Simonin. Formal Specification Approach of Role Dynamics in Agent Organisations: Application To the Satisfaction-Altruism Model. *J. Softw. Eng. Knowl. Eng.*, 17(05):615–641, 2007.

[90] M. Hinchey, J. Rash, C. Rouff, W. Truszkowski, and A. Vanderbilt. Requirements of an Integrated Formal Method for Intelligent Swarms. In *Form. Methods Ind. Crit. Syst.*, pages 33–59. Wiley-Blackwell, 2012.

[91] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[92] C. A. R. Hoare. Viewpoint: Retrospective: An Axiomatic Basis for Computer Programming. *Commun. ACM*, 52(10):30–32, 2009.

[93] R. Hoffmann, M. Ireland, A. Miller, G. Norman, and S. Veres. Autonomous agent behaviour modelled in PRISM – A case study. In *Model Checking Softw.*, volume 9641 of *LNCS*, pages 104–110. Springer, 2016.

[94] Y. Hua, S. Zander, M. Bordignon, and B. Hein. From AutomationML to ROS: A model-driven approach for software engineering of industrial robotics using ontological reasoning. In *Emerg. Technol. Fact. Autom.*, volume 21, pages 1–8. IEEE, 2016.

[95] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu. ROSRV: Runtime Verification for Robots. In *Runtime Verif.*, pages 247–254. 2014.

[96] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *Comput. Aided Verif.*, volume 10426 of *LNCS*, pages 3–29. Springer, 2017.

[97] M. U. Iftikhar and D. Weyns. A Case Study on Formal Verification of Self-Adaptive Behaviors in a Decentralized System. *Theor. Comput. Sci.*, 91:45–62, 2012.

[98] M. U. Iftikhar and D. Weyns. ActivFORMS: active formal models for self-adaptation. In *Softw. Eng. Adapt. Self-Managing Syst.*, pages 125–134. ACM, 2014.

[99] P. Izzo, H. Qu, and S. Veres. A stochastically verifiable autonomous control architecture with reasoning. In *Conf. Decis. Control*, pages 4985–4991, 2016.

[100] C. Jang, S. I. Lee, S. W. Jung, B. Song, R. Kim, S. Kim, and C. H. Lee. OPRoS: A New Component-Based Robot Software Platform. *ETRI J.*, 32(5):646–656, 2010.

[101] M. Kamali, L. Dennis, O. McAree, M. Fisher, and S. Veres. Formal verification of autonomous vehicle platooning. *Sci. Comput. Program.*, 148:88–106, 2017.

[102] M. Kamali, S. Linker, and M. Fisher. Modular verification of vehicle platooning with respect to decisions, space and time. In *Work. Formal Techniques for Safety-Critical Systems*, volume 1008 of *CCIS*, pages 18–36. Springer, 2018.

[103] A. Kane, O. Chowdhury, A. Datta, and P. Koopman. A case study on runtime monitoring of an autonomous research vehicle system. volume 9333 of *LNCS*, pages 102–117. Springer, 2015.

[104] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Conf. Decis. Control*, pages 2222–2229. IEEE, 2009.

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

[106] M. Kim, K. Kang, and H. Lee. Formal Verification of Robot Movements - a Case Study on Home Service Robot SHR100. In *Robot. Autom.*, pages 4739–4744. IEEE, 2005.

[107] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. 2007.

[108] M. Kloetzer, X. C. Ding, and C. Belta. Multi-robot deployment from LTL specifications with reduced communication. In *Decis. Control Eur. Control Conf.*, pages 4867–4872. IEEE, 2011.

[109] J. Knight, C. Dejong, M. Gibble, and L. Nakano. Why Are Formal Methods Not Used More Widely ? In *NASA Form. Methods*, page 12. Virginia Univ., 1997.

[110] S. Konur, C. Dixon, and M. Fisher. Formal Verification of Probabilistic Swarm Behaviours. In *Swarm Intell.*, volume 6234 of *LNCS*, pages 440–447. Springer, 2010.

[111] S. Konur, C. Dixon, and M. Fisher. Analysing robot swarm behaviour via probabilistic model checking. *Rob. Auton. Syst.*, 60(2):199–213, 2012.

[112] F. Kossak and A. Mashkoor. How to select the suitable formal method for an industrial application: A survey. volume 9675 of *LNCS*, pages 213–228. Springer, 2016.

[113] Y. Kouskoulas, D. Renshaw, A. Platzer, and P. Kazanzides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In *Hybrid Syst. Comput. Control*, page 263. ACM, 2013.

[114] P. Kouvaros and A. Lomuscio. Verifying fault-tolerance in parameterised multi-agent systems. In *Int. Jt. Conf. Artif. Intell.*, pages 288–294, 2017.

[115] H. Kress-Gazit, G. Fainekos, and G. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Robot. Autom.*, pages 3116–3121. IEEE, 2007.

[116] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu. Correct, Reactive, High-Level Robot Control. *Robot. Autom. Mag.*, 18(3):65–74, 2011.

[117] H. Liang, J. S. Dong, J. Sun, and W. E. Wong. Software monitoring through formal specification animation. *Innov. Syst. Softw. Eng.*, 5(4):231, 2009.

[118] W. Liu, A. Winfield, and J. Sa. Modelling Swarm Robotic Systems: A Case Study in Collective Foraging. In *Proc. Towar. Auton. Robot. Syst.*, pages 25–32, 2007.

[119] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Comput. aided Verif.*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

[120] Y. Lopes, S. Trenkwalder, A. Leal, T. Dodd, and R. Groß. Supervisory control theory applied to swarm robotics. *Swarm Intell.*, 10(1):65–97, 2016.

[121] D. Lyons, R. Arkin, S. Jiang, D. Harrington, F. Tang, and P. Tang. Probabilistic Verification of Multi-robot Missions in Uncertain Environments. In *Tools with Artif. Intell.*, pages 56–63. IEEE, 2015.

[122] M. Machin, F. Dufossé, J. Guiochet, D. Powell, M. Roy, and H. Waeselynck. Model-Checking and Game theory for Synthesis of Safety Rules. In *High Assur. Syst. Eng.*, pages 36–43. IEEE, 2015.

[123] M. Massink, M. Brambilla, D. Latella, M. Dorigo, and M. Birattari. On the use of Bio-PEPA for modelling and analysing collective behaviours in swarm robotics. *Swarm Intell.*, 7(2-3):201–228, 2013.

[124] T. Merz, P. Rudol, and M. Wzorek. Control System Framework for Autonomous Robots Based on Extended State Machines. In *Auton. Auton. Syst.*, pages 14–14, 2006.

[125] S. Mitsch, K. Ghorbal, and A. Platzer. On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles. In *Robot. Sci. Syst. IX*, page 8. Robotics: Science and Systems Foundation, 2013.

[126] S. Mitsch, K. Ghorbal, D. Vogelbacher, and A. Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *J. Rob. Res.*, 36(12):1312–1340, 2017.

[127] A. Miyazawa, P. Ribeiro, W. Li, A. Cavalcanti, and J. Timmis. Automatic property checking of robotic applications. In *Intell. Robot. Syst.*, pages 3869–3876. IEEE, 2017.

[128] S. Moarref and H. Kress-Gazit. Decentralized control of robotic swarms from high-level temporal logic specifications. In *Multi-Robot Multi-Agent Syst.*, pages 17–23. IEEE, 2017.

[129] L. Molnar and S. Veres. System verification of autonomous underwater vehicles by model checking. In *Ocean. 2009 - Eur.*, number 1, pages 1–10. IEEE, 2009.

[130] J. Morse, D. Araiza-Illan, J. Lawry, A. Richards, and K. Eder. Formal Specification and Analysis of Autonomous Systems under Partial Compliance. 2016.

[131] A. Nordmann, N. Hochgeschwender, D. Wigand, and S. Wrede. A Survey on Domain-Specific Languages in Robotics. *J. Softw. Eng. Robot.*, 7(July):75–99, 2016.

[132] M. O'Brien, R. Arkin, and D. Harrington. Automatic Verification of Autonomous Robot Missions. In *Simulation, Model. Program. Auton. Robot.*, volume 8810 of *LNCS*, pages 462–473. Springer, 2014.

[133] L. Pallottino, V. Scordio, A. Bicchi, and E. Frazzoli. Decentralized cooperative policy for conflict resolution in multivehicle systems. *Trans. Robot.*, 23(6):1170–1183, 2007.

[134] D. Phan, J. Yang, D. Ratasich, R. Grosu, S. Smolka, and S. Stoller. Collision avoidance for mobile robots with limited sensing and limited information about the environment. In *Runtime Verif.*, volume 9333 of *LNCS*, pages 201–215. Springer, 2015.

[135] A. Platzer. Differential Dynamic Logic for Verifying Parametric Hybrid Systems. In *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007.

[136] R. Podorozhny, S. Khurshid, D. Perry, and X. Zhang. Verification of Multi-agent Negotiations Using the Alloy Analyzer. In *Integr. Form. Methods*, volume 4591 of *LNCS*, pages 501–517. Springer, 2007.

[137] E. Prestes, J. L. Carbonera, S. R. Fiorini, V. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Gonçalves, M. Barreto, M. Habib, A. Chibani, S. Gérard, Y. Amirat, and C. Schlenoff. Towards a core ontology for robotics and automation. *Rob. Auton. Syst.*, 61(11):1193–1204, 2013.

[138] M. Proetzsch and K. Berns. Formal verification of safety behaviours of the outdoor robot ravon. In *Informatics Control. Autom. Robot.*, pages 157–164, 2007.

[139] M. Proetzsch, T. Luksch, and K. Berns. The behaviour-based control architecture iB2C for complex robotic systems. In *Annu. Conf. Artif. Intell.*, volume 4667 of *LNCS*, pages 494–497. Springer, 2007.

[140] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *Open Source Softw.*, volume 3, page 5. IEEE, 2009.

[141] V. Raman and H. Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP. In *Comput. Aided Verif.*, volume 6806 of *LNCS*, pages 663–668. Springer, 2011.

[142] A. Rao and M. Georgeff. BDI Agents: From Theory to Practice. In *Multi-Agent Syst.*, pages 312–319, 1995.

[143] G. Reed and A. Roscoe. A timed model for communicating sequential processes. 226(1-3):314–323, 1986.

[144] P. Ribeiro, A. Miyazawa, W. Li, A. Cavalcanti, and J. Timmis. Modelling and verification of timed robotic controllers. In *Integr. Form. Methods*, volume 10510 of *LNCS*, pages 18–33. Springer, 2017.

[145] J. O. Ringert, A. Roth, B. Rumpe, and A. Wortmann. Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems. *J. Softw. Eng. Robot.*, 6(1):33–57, 2015.

[146] M. Risler and O. von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in XABSL. In *Form. Model. Methods Multi-Robot Syst.* Citeseer, 2008.

[147] A. Rizaldi, J. Keinholz, M. Huber, J. Feldle, F. Immler, M. Althoff, E. Hilgendorf, and T. Nipkow. Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/HOL. volume 10510 of *LNCS*, pages 50–66. Springer, 2017.

[148] G. Rosu and K. Havelund. Rewriting-Based Techniques for Runtime Verification. *Autom. Softw. Eng.*, 12(2):151–197, 2005.

[149] C. Rouff, W. Truszkowski, J. Rash, and M. Hinchey. A Survey of Formal Methods for Intelligent Swarms. Technical report, SAIC and NASA, 2004.

[150] C. Rouff, A. Vanderbilt, W. Truszkowski, J. Rash, and M. Hinchey. Formal Methods for Autonomic and Swarm-based Systems. In *Leveraging Appl. Form. Methods*, pages 100–102, 2004.

[151] C. Schlegel and R. Worz. The software framework SMARTSOFT for implementing sensorimotor systems. In *Intell. Robot. Syst.*, volume 3, pages 1610–1616. IEEE, 1999.

[152] B. Schoettle and M. Sivak. A survey of public opinion about connected vehicles in the U.S., the U.K., and Australia. In *Int. Conf. Connect. Veh.*, pages 687–692. IEEE, 2014.

[153] D. Selsam, P. Liang, and D. L. Dill. Developing bug-free machine learning systems with formal mathematics. In *Int. Conf. Machine Learning*, volume 70, pages 3047–3056. JMLR, 2017.

[154] D. Seto, B. Krogh, L. Sha, and A. Chutinan. The Simplex architecture for safe online control system upgrades. In *Am. Control Conf.*, volume 6, pages 3504–3508. IEEE, 1998.

[155] A. Shakhimardanov, N. Hochgeschwender, and G. Kraetzschmar. Component models in robotics software. In *Perform. Metrics Intell. Syst.*, page 82. ACM, 2010.

[156] R. Simmons, C. Pecheur, and G. Srinivasan. Towards automatic verification of autonomous systems. In *Intell. Robot. Syst.*, volume 2, pages 1410–1415. IEEE, 2000.

[157] O. Simonin and J. Ferber. Modeling Self Satisfaction and Altruism to handle Action Selection and Reactive Cooperation. *Simul. Adapt. Behav. From Anim. to Animat.*, pages 314–323, 2000.

[158] P. Sistla, M. Zefran, and Y. Feng. Runtime monitoring of stochastic cyber-physical systems with hybrid state. In *Runtime Verif.*, volume 7186 of *LNCS*, pages 276–293. Springer, 2012.

[159] G. Smith and Q. Li. Maze: An extension of object-z for multi-agent systems. In *Abstr. State Mach. Alloy. B, TLA, VDM, Z*, volume 8477 of *LNCS*, pages 72–85. Springer, 2014.

[160] T. Sotiropoulos, H. Waeselynck, J. Guiochet, and F. Ingrand. Can robot navigation bugs be found in simulation? An exploratory study. In *Softw. Qual. Reliab. Secur.*, pages 150–159. IEEE, 2017.

[161] D. Stampfer. The SmartMDSD Toolchain : An Integrated MDSD Workflow and IDE for Robotics Software. *J. Softw. Eng. Robot.*, 1(July):3–19, 2016.

[162] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merk. *Eclipse Modeling Framework*. Addison-Wesley, 2008.

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

[164] K. Talamadupula, G. Briggs, T. Chakraborti, M. Scheutz, and S. Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *Conf. Intell. Robot. Syst.*, pages 2957–2962. IEEE, 2014.

[165] A. Tarasyuk, I. Pereverzeva, E. Troubitsyna, T. Latvala, and L. Nummila. Formal development and assessment of a reconfigurable on-board satellite system. volume 7612 of *LNCS*, pages 210–222. Springer, 2012.

[166] M. Tenorth and M. Beetz. KnowRob—knowledge processing for autonomous personal robots. In *Intell. Robot. Syst.*, pages 4261–4266. IEEE, 2009.

[167] S. Tigane, L. Kahloul, and S. Bourekkache. Reconfigurable stochastic petri nets for reconfigurable manufacturing systems. In *Int. Work. Service Orientation in Holonic and Multi-Agent Manufacturing*, volume 694 of *SCI*, pages 383–391. Springer, 2016.

[168] A. Tiwari. Abstractions for Hybrid Systems. *Form. Methods Syst. Des.*, 32:57–83, 2008.

[169] M. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. In *Logics Concurr. - Struct. versus Autom.*, volume 1043 of *LNCS*, pages 238–266. Springer, 1996.

[170] W. Visser, K. Havelund, G. Brat, S. J. Park, and F. Lerda. Model Checking Programs. *Autom. Softw. Eng.*, 10(2):3–11, 2002.

[171] M. Wächter, S. Ottenhaus, M. Kröhnert, N. Vahrenkamp, and T. Asfour. The ArmarX Statechart Concept: Graphical Programing of Robot Behavior. *Front. Robot. AI*, 3:33, 2016.

[172] D. Walter, H. Täubig, and C. Lüth. Experiences in applying formal verification in robotics. volume 6351 of *LNCS*, pages 347–360. Springer, 2010.

[173] M. Webster, N. Cameron, M. Fisher, and M. Jump. Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. *J. Aerosp. Inf. Syst.*, 11(5):1–31, 2014.

[174] M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. L. Koay, and K. Dautenhahn. Formal Verification of an Autonomous Personal Robotic Assistant. *AAAI FVHMS*, pages 74–79, 2014.

[175] M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. L. Koay, K. Dautenhahn, and J. Saez-Pons. Toward Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study. *Trans. Human-Machine Syst.*, 46(2):186–196, 2016.

[176] M. Webster, M. Fisher, N. Cameron, and M. Jump. Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems. volume 6894 of *LNCS*, pages 228–242. Springer, 2011.

[177] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. A Survey of Formal Methods in Self-adaptive Systems. In *Comput. Sci. Softw. Eng.*, C3S2E '12, pages 67–79. ACM, 2012.

[178] D. Weyns, S. Malek, and J. Andersson. FORMS: a FOrmal Reference Model for Self-adaptation. In *Auton. Comput.*, page 205. ACM, 2010.

[179] D. Weyns, S. Malek, and J. Andersson. FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *Trans. Auton. Adapt. Syst.*, 7(1):8, 2012.

[180] J. Whittle, P. Sawyer, N. Bencomo, B. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *Requir. Eng. Conf.*, pages 79–88. IEEE, 2009.

[181] A. Winfield and M. Jirotka. The case for an ethical black box. volume 10454 of *LNAI*, pages 262–273. Springer, jul 2017.

[182] A. Winfield, W. Liu, J. Nembrini, and A. Martinoli. Modelling a wireless connected swarm of mobile robots. *Swarm Intell.*, 2(2-4):241–266, 2008.

[183] A. Winfield and J. Nembrini. Safety in numbers: fault-tolerance in robot swarms. *J. Model. Identif. Control*, 1(1):30, 2006.

[184] A. Winfield, J. Sa, M.-C. Fernández-Gago, C. Dixon, M. Fisher, M. C. F. Gago, C. Dixon, and M. Fisher. On Formal Specification of Emergent Behaviours in Swarm Robotic Systems. *J. Adv. Robot. Syst.*, 2(4):363–370, 2005.

[185] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal methods: Practice and Experience. *ACM Comput. Surv.*, 41(4):1–36, 2009.

[186] 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 *Auton. Agents Multiagent Syst.*, volume 23 of *AAMAS*, pages 79–86, 2008.

[187] E. Şahin. Swarm Robotics: From Sources of Inspiration to Domains of Application. In *Swarm Robot.*, pages 10–20. Springer, 2005.