The Need for a Meta-Architecture for Robot Autonomy

Stalin Muñoz Gutiérrez sgutierr@ist.tugraz.at
Gerald Steinbauer-Wagner steinbauer@ist.tugraz.at
Autonomous Intelligent Systems Group.
Institute of Software Technology. Graz University of Technology. Austria.

Long-term autonomy of robotic systems implicitly requires dependable platforms that are able to naturally handle hardware and software faults, problems in behaviors, or lack of knowledge. Model-based dependable platforms additionally require the application of rigorous methodologies during the system development, including the use of correct-by-construction techniques to implement robot behaviors. As the level of autonomy in robots increases, so do the cost of offering guarantees about the dependability of the system. Certifiable dependability of autonomous robots, we argue, can benefit from formal models of the integration of several cognitive functions, knowledge processing, reasoning, and meta-reasoning. Here, we put forward the case for a generative model of cognitive architectures for autonomous robotic agents that subscribes to the principles of model-based engineering and certifiable dependability, autonomic computing, and knowledge-enabled robotics.

1 Introduction

It can be argued that *acting* is the crux of robotics research. It is the agency of robots manifested as continuous interaction with a physical world that distinguishes the robotics field from other scientific and technological disciplines. Because robots are designed by humans with the help of computers to fulfill a purpose, they extend our agency. The quest for robot autonomy is bound to the quest for ours. We ponder the implications of intelligent robots able to make not only decisions but reason about their own goals and intentions. In particular, for safety-critical applications we would like to have guarantees about the behavior of autonomous robots. Research on the dependability of engineering solutions has a long tradition that has produced a plethora of methodologies, techniques, technologies, standards, and regulations. We expect from our robots the same we expect from other machines, devices, and tools: to satisfy the highest quality standards at a minimum cost. If we are to see autonomous robots potentiate our agency in the world, we should aim to make them certifiable. However, to many researchers, the mere mentioning of words such as standards and certification brings the preconception that these aspects do not qualify as research activities. Whether this might be the case when seen from the perspective of specialized research areas, we argue that this judgment does not apply for integrative multidisciplinary endeavors such as *long-term robot autonomy* (LTRA). Community efforts to come to an agreement on standard models and platforms pay in the long run better than scattered unconcerted attempts to tackle a problem as big as that of LTRA.

A wide diversity of research interests is currently driving the robotics community in an expanding effort to include more aspects of intelligence to enable autonomy in robots. Nevertheless, integrating autonomous behavior out of a diverse collection of paradigms, formal methods, models, technologies, and methodologies is a heuristic, often trial and error process that does not scale well to complex domains. Reference architectures are helpful, provided they can be effortlessly adapted by roboticists to achieve different levels of autonomy. In practice, however, implementing a very abstract architecture requires of many design decisions and significant human programming effort prone to interpretation and

R.C. Cardoso, A. Ferrando, F. Papacchini,
M. Askarpour, L.A. Dennis (Eds.): Second
Workshop on Agents and Robots for reliable
Engineered Autonomy (AREA'22).
EPTCS 362, 2022, pp. 81–97, doi:10.4204/EPTCS.362.9

implementation defects. More importantly, research within the field of robotics should not be primarily a software engineering effort that do not create novel theories nor ask fundamental scientific questions.

We agree with the view that *Model-based engineering* (MBE) can reliably support the development of highly complex systems that require of safety guarantees. The maturity and availability of formal methods, modeling languages, and automation toolchains within the MBE field are no longer limiting factors for a paradigm shift from hand-crafted robotic systems to correct-by-construction affordable certifiable solutions. In the long run, the research community would likely benefit from interoperability enabled by MBE models. Models that would not only be executable facilitating their use by practitioners, but also amenable to be easily reproduced, analyzed, compared, and improved by any research group that wishes to do so.

The contribution of our work is an integrative literature review that spans different concerns relevant to LTRA. We put forward the case for a generative model of cognitive architectures, that is, a formal description captured in one or more *domain specific modeling languages* (DSMLs) enabling automatic synthesis of concrete architectures for LTRA as a satisfying solution to a set of traceable requirements. We will highlight the need to further study the integration of several deliberation functions besides acting, that are necessary for intelligent behavior. In addition, we consider that cognitive penetrability and meta-reasoning models should be explicit and formal, amenable to theoretical analysis, verification, and automatic transformations. In the next section, we comment on relevant research on dependability, model-driven engineering, autonomic computing, autonomy, and robot architectures. Section 3 will present the case for a meta-architecture for autonomy in robots. Finally, in Section 4, we present our concluding remarks and future work.

2 Related Work

We start our presentation by highlighting the relevant qualities of dependable systems because we consider dependability to be concomitant to LTRA. In our account, we intentionally left out many core aspects that apply to almost all engineering application domains such as fault tolerance, reconfiguration, graceful degradation, and so on. Instead, in the interest of conciseness, we will focus our presentation on deliberative and high-level cognitive functions.

2.1 Dependability

Traditional studies on the conceptualization of dependable information systems include the works of Laprie [27] and Aviẑienis et al. [3]. Their concepts and taxonomies cover a wide range of engineering concerns to take into consideration when defining a generic cognitive architecture that aims for dependability. For Aviẑienis et al., dependability is a *global concept* encompassing several attributes: availability, reliability, safety, integrity and maintainability. Closely related concepts seeking equivalent goals are dependability or the "ability to deliver service that can justifiably be trusted [and]... to avoid [unacceptable] system failures", *high-confidence* where "consequences of the system behavior are well understood and predictable", *survivability* or the "capacity of a system to fulfill its mission in a timely manner", and *trustworthiness* or the "assurance that a system will behave as expected". According to their study, dependability can be engineered by means of four mechanisms: (1) *fault prevention*, (2) *fault tolerance*, (3) *fault removal*, and (4) *fault forecasting*. They highlight the importance of *self-checking*, implying that the mechanisms responsible for handling faults, need to be themselves protected from the impact of the potential faults.

The European SPARC 2020 Robotics Multi-Annual Roadmap [44] (MAR) proposes the following levels of dependability numbered from 0 to 7 in increasing order: *no dependability*, *mean failure dependability*, *fails safe*, *failure recovery*, *graceful degradation*, *task dependability*, *mission dependability*, and *predictive dependability*. A generative cognitive architecture would need to address all levels above level 2 as far as the robot behavior is concerned. The roadmap also identifies *cognitive dependability* as significant for robots that require understanding, interpretation, and reasoned decision making.

Dependability of robotic systems needs to be *certified*. A certificate resolves that the deployment of an autonomous system cannot incur in undesirable consequences nor lead to unacceptable risks based on pondering *claims* and *evidence*, and *argumenting* in favor of the veracity of the claims, given the evidence [40]. The *Committee on Certifiably Dependable Sofware Systems* (CCDSS) of the National Research Council of the National Academies [23] defines certification as "the process of assuring that a product or process has certain stated properties, which are then recorded in a certificate." To certify a software as dependable they consider it should be done based on a "credible" *dependability case* [55] (a generalization of the concept of *safety case* [7]). The required level of dependability may significantly constrain the architecture, models and technologies chosen for its implementation. Some additional key points made by the committee are that developing software for dependable systems is difficult and costly and that reliance on testing does not constitute sufficient evidence if high dependability is required (evidence from analysis is also necessary). About the last point, extensive testing is nevertheless mandatory. Unit tests, fuzz testing (randomly generated tests), automatic test generation from models such as state machines (model-based testing), invariants and run-time assertions , as well as performing automatic tests after every maintenance action constitute effective techniques towards achieving dependability requirements. Some of the existing standards are: (1) *Common Criteria* (CC), a security certification standard for software; (2) *Software Considerations in Airborne Systems and Equipment Certification* (RTCA DO-178B)(in Europe is known as ED-12B), a worldwide standard for software in civilian aircraft, and (3) International Electrotechnical Commission's (IEC's) 61508 (in US its counterpart is known as ISA S84.01), a standard for functional safety of electrical/electronic/programmable electronic-safety-related systems.

Addressing all relevant aspects of dependable systems is beyond the scope of this presentation, it requires processes that span beyond the strictly technical realm. For example, CCDSS points out the need to evaluate a dependability claim beyond the technical dimension. It is also necessary to evaluate the organization that stated the claim, to assess the integrity of the evidence chain. As for the importance of a *safety culture*, the committee regards it as important to grant organizational support to personnel attitudes aiming for the highest quality standards. In [48], Steinbauer et al. propose a *Model-based Development Process* (MBDP) for dependable autonomous systems. This model covers the whole lifecycle of the system as well as different aspects of the system architecture. A comprehensive treatment of this approach including how to deal with principal threats to dependable systems can be found in [47].

2.2 Model Based Engineering

Engineering solutions benefit extensively from the use of models to synthesize solutions to real-world problems faced by society. Within the field of software engineering, modeling has been there since the beginning. Recently, the term *Model-Driven Engineering* (MDE) has emerged as a promising approach that can effectively cope with the increasing complexity of software systems. In [41], Schmidt presents a clear perspective on why MDE is, at least in intent and focus, different in the role that models play within the field. The main distinction is that most abstractions and traditional models have primarily been *computing-oriented* as opposed to *domain-oriented*. He interprets the limited adoption of *Computer-*

Aided Software Engineering (CASE) methods by the software engineering community as an example that illustrates that models by themselves are not sufficient. According to his assessment, one of the reasons for the arguable failure of CASE is that the models on which CASE relied were too abstract, with a *"one-size-fits-all"* perspective that created enormous challenges for automatic translation to particular system platforms, therefore creating big *semantic gaps* between the design intent and its often intricate realization using specific technologies. To address this problem, Schmidt proposes developing MDE technologies by combining two approaches: (1) *Domain Specific Modeling Languages* (DSMLs), and (2) transformation engines and generators. DSMLs are meta-models tailored for the application domain. The purpose of DSMLs is to allow developers to express design intent *declaratively* and not *imperatively*. The purpose of transformation engines and generators is to ensure consistency with functional and non-functional requirements. Additionally, MDE has to deal with the orchestration and customization of product-line architectures, model checkers, aspect-oriented languages, design patterns, etc.

In [24], Kent discusses the benefits of a *model-centric* approach for architecting engineering solutions. A primary reason for having an architectural description is to be able to communicate effectively with different human stakeholders. Models abstract away platform-specific concerns. From the perspective of the system development, they allow for manipulation and transformation by machines facilitating automatic validation of correctness and model-driven testing. He reflects on the potential of *meta-models*, envisioning their use to drive a set of automated tools where tools can generate other tools and reconfigure themselves.

Steck et al. [46] applied MDE to the development lifecycle of service robot architectures. In their work, they define robotic meta-models for software components. Models then ease their transformation into executable code and are used for various analyses and simulations at design-time, and support decision-making at run-time. Models are a means to describe the features of components, allowing the application of Component-Based Software Engineering (CBSE) principles [8, 9]. The central element of their architecture is a *model pool* that contains various models with heterogeneous representations required for the particular technological platforms and component implementations. Meta information makes it possible to query information, retrieve them in the target representation or make it possible to reason about them. Their toolchain is SMARTSOFT [2], a robotic framework that instantiates their platform independent meta-model SMARTMARS.

In [45], Stampfer et al. present SmartMDSD toolchain, an integrated model-driven software development environment (DE) for robotics software development based on SMARTSOFT. One of the main features of SmartMDSD is that it integrates several DSMLs into a single consistent DE. Its design is based on finding the right trade-off between *freedom of choice* and *freedom from choice*. The former aims not to enforce any decisions while the latter provides sufficient guidance regarding the composability and system-level conformance. Additional embraced design principles are *separation of roles*, *separation of concerns*, and *composability and composition*. RobMoSys [16] is a MDE approach to robotics development. It relies in meta-models incorporating Papyrus [38] for Robotics and SmartMDSD. RobMoSys tackles complexity by considering separation of concerns: (1) computation, (2) communication, (3) coordination, and (4) configuration. Concerns apply across different levels of abstraction: (1) hardware, (2) OS System/Middleware, (3) execution container, (4) function, (5) service, (6) skill, (7) task plot, and (8) mission. The project aims for robot technologies to be composable, modular, predictable, safe, reusable, and certified. An integrated feature of Papyrus is the possibility of automatically generating hazard and risk analysis. There are currently several robotic platforms following MDE principles, a recent survey can be found in [14].

2.3 Autonomic Computing

One example of the increasing importance of self-managing computer systems is the IBM *autonomic computing initiative* [18]. The term autonomic was chosen based on the autonomic nervous system in animals that releases the conscious brain from duties that are vital (heart rate, respiratory rate, digestion, etc.) but monotonous. The initiative supports the case for the development of systems that are: (1) *self-configuring*, systems automatically adapt to dynamically changing environments, new features and capabilities can be added/subtracted/enabled/disabled; (2) *self-healing*, systems detect, diagnose, and react to anomalies, they predict problems and take preventive actions, their design maximizes reliability and availability; (3) *self-optimizing*, the system can monitor and adapt resources automatically optimizing across multiple heterogeneous systems; and (4) *self protecting*, they anticipate, detect, identify, and protect themselves from attacks. The report also calls for the widespread use of open industry standards.

As pointed out by Garlang et al. [19] the need to cope with changing environments points towards systems that are capable of dynamic *self-adaption* by means of external mechanisms. He remarks that although self-adaptation is being in computer systems for a while, the mechanisms are often specific to the application, the code has no separation of concerns and exhibits only localized handling of faults. External self-adaptation mechanisms subscribe to proper software engineering principles where separation of concerns and loose coupling allow for modules than facilitate analysis, maintenance, modification, and reuse by new or existing modules or systems. There is a case for *architectural models* (AMs) to enable reasoning about self-adaptation for satisfying system-level behaviors and properties, as well as integrity constraints. They proposed the *Rainbow* framework, an abstract AM and reusable infrastructure to support self-adaptation. The framework monitors the properties of a running system, checking for constraint violations against the AM that consists of a graph made of *components* (computational elements) linked by *connectors* (their interactions). Rainbow models the system adaptation by means of an extended *architectural style* that includes the notion of *adaptation operators* (AOs) and *adaptation strategies* (ASs). AOs (for example adding or removing a service) are possible actions to perform by the control infrastructure to change the configuration of the system. ASs are adaptations necessary to avoid an undesirable condition (for example to activate a graceful degradation in the services).

Oreizy et al. [35] defines software as *self-adaptive* if it modifies its own behavior in response to changes in the operating environment. In deciding the self-adaptation capabilities of a system, several concerns must be taken into consideration: (1) triggering conditions; (2) open or close adaptation, i.e. it is open if the system can incorporate new behaviors while running, it is close otherwise; (3) type or level of autonomy; (4) cost-effectiveness of adaptation; (5) frequency of adaptation; and (6) information required for deciding on triggering adaptation. According to the *spectrum* of self-adaptability proposed by them, highly adaptive solutions will exhibit a clear separation of software-adaptation concerns from other software functions. Safety, reliability, and correctness are primordial for self-adaptive systems design. It is also important to guarantee the consistency and integrity of the system. Oreizy et al. propose the use of an *Architecture Evolution Manager* (AEM) that mediates all adaptations. The AEM enforces that all changes are *atomic*, meaning they should not fail, and if they do so they should not be deployed, leaving the application unaltered. Another fundamental architectural component in their proposal is the Adaptation Management (AM). AM is a monitor of the application and its environment. It enables adaptations, plans them, and deploys change descriptions to the running system. These functions can be implemented using independent software agents. For decision making, observations are gathered from exceptional events generated by *inline observers* that check relevant assertions. Monitoring of *patterns* of events may be necessary to trigger an adaptation, this can be implemented using expectation agents. Consistency Management (CM) can check *invariants* offline on *attributed graph grammars* that capture

all possible configurations for the application, in this representation graph rewrites represent architectural changes. Observers performing runtime monitoring on the application and the environment are employed to detect inconsistencies against relevant *annotations*.

2.4 Autonomy

Franklin and Graesser [17] discuss the attributes of autonomous agents and provide a general definition that applies to solutions and systems within the field of robotics: "An *autonomous* agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." As pointed out by the proponents, this embracing definition equally comprises complex sophisticated cognitive agents as it does a simple closed-loop controlled device such as a thermostat. They identify additional capabilities that are more suitable for an operational definition in the context of decision making, the core features are reactivity, autonomy, goal orientation, and temporal continuity. In the context of robotics research, Tessier [50] quotes the definition of autonomy considered by the Defense Science Board as "the capability to independently compose and select among different courses of action to accomplish goals based on its knowledge and understanding of the world, itself, and the situation" [13] (Originally from Shattuck [42]). Although highly relevant to the discussion, and in particular to clarify why a concept of *dependable autonomy* is not a *contradictio in terminis*, we are not addressing here Decisional Autonomy from the perspective of *shared authority* and the broader view that autonomy can be understood as a power relationship between human and robot agencies. Module this abstraction, autonomy is, in words of Castelfranchi and Falcone, "due to agent's architecture" [10]. The argument for a cognitive architecture is therefore of necessity not of sufficiency.

Decisional autonomy is an important topic covered in the MAR. The level of autonomy of a robot is a consequence of its cognitive abilities but it is also parameterized by: *environmental factors* modulating its empowerment, *decision cost* demanding risk-aware well informed choices, the *time scale* of deployments demanding high dependability, and the extent of the *decision range* making the required level of autonomy more complex is it widens. Hereafter when discussing autonomy we omit the adjective *long-term*, as this can be understood as a cognitive ability parameter that demands the robot a higher level of dependability.

2.5 Robot Architectures

We acknowledge that robot architectures are diverse and accept numerous variations. Our intention here is not to give a comprehensive account nor to put forward a new taxonomy. We bring into attention some fundamental concepts and a handful of reference architectures chosen because of their prominence and their influence in the research community. We intend to bring to the table, through these exemplars, the concepts that have proven essential and operational to real-world robotic solutions. It should not be implied that other works are less influential or important.

2.5.1 Software Engineering

Software engineering theory and good practices of architectural models are structural to the equation. The realization of any paradigm, theory, behavior, or experimental study in robotics requires a software engineering dimension. Architectural descriptions *are models* described in architectural description languages (ADLs). In practice, architectural descriptions go all the range from just a collection of connected

boxes without underlying semantics, to fully formal models.

In [30], Medvidovic and Taylor identify core principles of software architectures and provide a conceptual framework that we briefly summarize here. Architectures focus on the high-level structure of a software application. ADLs allow the explicit modeling of components, connectors, configurations, constraints, hierarchical composition, computational paradigms, and communication paradigms among others. The first concern of ADLs is that of expressivity and *representation*. They also must assist the whole design process, facilitate static and dynamic analysis, and address evolution, refinement, traceability, and simulation. Toolsets come together with an ADL, their adoption depends strongly on the availability of a flexible and powerful set of technologies that facilitate the realization of the architecture during the whole life-cycle of development. One important aspect to consider when selecting an ADL beyond its concrete syntax is the semantic theory that subsumes the semantic framework that comes with it. Some of these theories are: statecharts, finite state machines, partially-ordered event sets, chemical abstract machines, process algebras, Petri nets, algebraic or axiomatic formalisms. The adherence to semantic theories facilitates the automatic tools to perform model transformations, analysis, evaluation, and verification for example. Three fundamental elements of software architectures are (1) components, (2) connectors, and (3) architectural configurations. For the first two elements, the ADL will allow its formal definition, and specification of interfaces, types, semantics, constraints, evolution, and the so-called non-functional properties among which more dependability aspects are expressed (safety, and reliability for example). Regarding architectural configurations, it includes the definition of a *topology*, a graph of components and connectors representing the architectural structure in a way that allows for the assessment of concurrent and distributed aspects. The qualities of the configuration description are understandability, compositionality, refinement, traceability, and heterogeneity. Qualities of the described system are heterogeneity, scalability, evolvability, and dynamism. Just to highlight one aspect, dynamism considers the modifications to the architecture where the system is running, this is important for safety- and mission-critical robotic systems.

ROS [37] has become the default choice for implementing software architectures in robotic solutions. In a recent study, Malavolta et. al. [29] analyse the adoption of ROS by the robotics community. In their study, they report on the use of ROS by practitioners, the top architecturally supported capabilities found are control, planning, vision, and navigation. The top quality requirements are maintainability, reliability, performance, and usability. A set of 49 architectural guidelines based on evidence were identified by their study.

2.5.2 Autonomy

Robot architectures specialized for autonomy have also been successfully implemented. In [1], Alami et al. present an integrated architecture consisting of three levels: *decision*, *execution*, and *functional*. They consider that autonomous robots must exhibit deliberation and reactive capabilities, broadly encompassing: (1) decision-making, (2) reactivity, (3) situation anticipation and (4) context-awareness. In their architecture, the decision level is both goal- and event-driven and includes automated planning, plan execution supervision, and responsiveness to events generated in other layers. The execution level, also known as *executive* is responsible for controlling and coordinating the execution of functions while satisfying the requirements of the task at hand. The functional level corresponds with the built-in robot action and perception capabilities. For them, deliberation is an important faculty, they define it as "both a goal-oriented process wherein the robot anticipates its actions and the evolution of the world, and also, a time-bounded, context-dependent decision-making process for a timely response to events." Their implementation of the architecture includes the use of the temporal planner IxTeT, the Procedural system

for task refinement and supervision PRS, the reactive control of the functional level Kheops, and GenoM for the specification and integration of modules at the functional level. PRS is a sophisticated implementation of the Belief-Desired-Intention paradigm as well as an expressive formal language that is a trade-off between declarative and imperative programming [21]. GenoM is a Model-Driven Software Engineering Tool that automatically generates the module's code as much as possible. A description language allows the definition of a given module capturing: (1) the services that it manages, (2) the input and output parameters, (3) associated execution code fragments and their real-time characteristics (time period and delay, for example), and (4) an exhaustive list of possible execution reports (feedback on success or failure of a module execution). Alami et al. architecture for autonomy has remained a reference architecture among the robotics research community. In a recent work [20], Hladik et al. present a Model-based execution engine for the control and verification of real-time systems. This engine developed at LAAS-CNRS is part of a *toolchain* called *HIPPO* that supports a unified modeling language for design, verification, and execution. HIPPO allows for the automatic generation of both executable and verifiable code from a high-level specification based on the robotic programming framework $G^{en}{}_{o}M$. The generated code is interpreted by a dedicated *real-time* execution engine.

Another archetypical example is the *Remote Agent* (RA) architecture for autonomous spacecrafts of NASA of Muscettola et al. [32]. Remote autonomous agents require of the following characteristics: (1) computational intelligence that exhibit high-levels of autonomy and continuous operation over long periods of time, (2) guarantees of meeting tight deadlines and resource constraints, (3) high reliability under partial observability and in the event of faults, and (4) precise coordination of concurrent processes over several subsystems. The three principles implemented in RA architecture are: (1) the use of *compositional declarative models* and *model-based programming*, (2) *on-board deduction and search*, and (3) *high-level closed-loop control*. Rather than relying on simplifying assumptions and abstractions, or depending on human supervision and decision-making, RA uses declarative models for the synthesis of appropriate responses to anomalies and unexpected scenarios. RA performs searches in a highly *tuned*, propositional, best-first kernel. Their high-level control design allows the declarative definition of missions, instead of the traditional concrete low-level commands defined for nonautonomous operations. There are four main architectural components of RA: (1) The *Reactive Executive* (EXEC), (2) The temporal *Planner/Scheduler* (PS), (3) the *Mission Manager* (MM), and (4) the *Model-based mode Identification and Reconfiguration* (MIR) module.

Here we highlight only what we consider main dependability dimensions addressed by MIR. In [56], Williams and Nayak further elaborates on the MIR module, presenting *Livingstone* the kernel for an autonomous *model-based* diagnostic system. Livingstone uses fast propositional conflict based algorithms for model-based diagnosis combined with a propositional, conflict-based feedback controller. Their work relies in a central *single model* for a variety of autonomous tasks including detecting anomalies, performing diagnosis, fault recovery, and fault avoidance. Components of the system are modeled using concurrent transition systems specified using temporal logic. Models are qualitative by choice, aiming to minimize detailed modeling mistakes that may lead to erroneous diagnoses. To achieve reactivity, their system limits reasoning to consider only the present and next state of the system. In [33], Nayak et al. provides with a detailed report on the formal verification of RA. The verification included nominal operation, goal-oriented commands, closed-looped plan execution, failure diagnosis, recovery, replanning, as well as system-level fault protection. Because the testing of RA autonomous operation would require of an impractically large number of test cases, and also due to the limited availability of high-fidelity test beds that were unable to run faster than real time, a *baseline testing* approach was conducted combined with the use of *lower fidelity testbeds* resenbling a *pyramid*.

2.5.3 Cognition

In [39], Reiter states that "Cognitive Robotics has, as its long-term objectives, the provision of a uniform theoretical and implementation framework for autonomous robotic or software agents that reason, act and perceive in changing, incompletely known, unpredictable environments."

In a recent survey [25], Kotseruba and Tsotsos review over 40 years of research in cognitive architectures identifying 84 of them with over different 900 projects. They provide a classification based on knowledge representation (KR) and information processing that divides them into *emergent* (E) and *hybrid* (H). Emergent architectures are based on massively parallel models, within this group, a further classification divides them into *connectionist logic systems* (CLS) and *neural modeling* (NM). Hybrid architectures combine symbolic and sub-symbolic approaches. A further refinement of this category divides them into *fully integrated* (FI), *symbolic sub-processing* (SSP), and purely *symbolic* (S). The most cited works include ACT-R (in H,FI), Soar (in H,FI), CLARION (in H,FI), ICARUS (in H,S), EPIC (in H,S), and LIDA (in H,FI). The previously presented architecture PRS is also classified as symbolic (in H,S). Discussing the particulars of these cognitive architectures is out of the scope of this integrative review.

Higher cognitive abilities enable higher levels of autonomy. MAR identifies the following cognitive abilities for robotics applications: (1) action, mainly concerned with planning and acting; (2) interpretative, or the ability to make sense of the environment and other agents; (3) envisioning, or the ability to understand the consequences of its actions and relevant events occurring in the environment; (4) learning, concerned with knowledge acquisition and (5) reasoning, or the capacity to process knowledge to make rational conclusions and choices.

In [54], Wang et al. present perspectives on *cognitive informatics* and *cognitive computing* (CC). In particular, they report on CC as an emergent paradigm aiming to develop *cognitive computers*[36] with the capacity to think and learn applicable to autonomous agent systems. They consider cognitive computers as "as knowledge processors beyond those of data processors in conventional computing". These computers would enable the simulation of *machinable thought* that consists on several reasoning modes: inferences, problem-solving, decision making, etc. They identify as a major challenge towards the realization of cognitive computers the handling of inconsistent information. The concept of *cognitive penetrability*[36] relates closely to the plasticity of the human cognitive process: "what we know determines how we react". The problem of modeling how to overcome inconsistencies is interesting not only because of the necessary adequate handling of them, but because according to cognitive science they may trigger important learning processes. For example, Zhang and Grégoire [57] describe that "through the process of overcoming an inconsistency, the agent is able to revise, refine, or augment its existing knowledge to adapt to the emergent patterns and behaviors as exhibited in the inconsistent circumstance". Their work also reports on several effective techniques for handling inconsistency in intelligent systems.

Beetz et al. [5] develop the Cognitive Robot Abstract Machine (CRAM), a software platform for the development of *cognition-enabled* autonomous robots. They identified the need for software tools for the implementation of high-level cognitive abilities such as learning and knowledge processing. The CRAM kernel includes CRAM plan language(CPL) specially designed for mobile manipulation and cognition-enabled control. The design principle is that control programs are not reduced to code artifacts but also exist as entities that allow reasoning on them. Another element of the kernel is *KnowRob* a knowledge processing system based on the formalism of description logic [4, 52] (see Section 2.5.4). On top of their kernel, there is *COGITO*, a meta-reasoning component that reasons and deals with plan execution flaws. *Cognitive extensions* (CRAM-EM) allow for: (1) action awareness, (2) learning and adaptation, (3) transformational planning, and (4) web-enabled robot control.

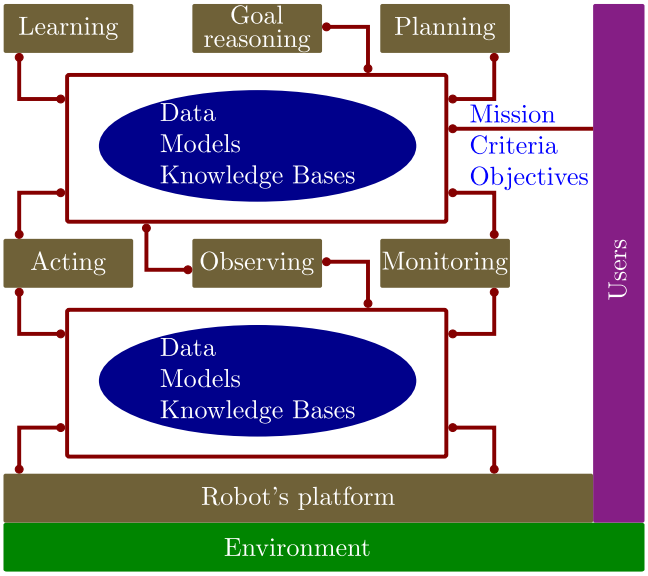

Figure 1: Depiction of a typical hierarchical architecture implementing several deliberation functions (modified from [22]). At the level of deliberation, the robot platform encapsulates and abstracts away the complexity of sensory-motor functions of the robot. More than one knowledge base might be required across different levels of abstraction.

In [22], Ingrand and Ghallab survey deliberation for autonomous robots. In their review, six deliberation functions are relevant for autonomous robots: (1) planning, (2) acting, (3) monitoring, (4) observing, (5) goal-reasoning, and (6) learning. The importance of deliberation resides in its expression as deliberative acts or acting towards intended objectives. They acknowledge the importance of bounded rationality and the need for meta-reasoning functions that trade deliberation time for action time according to the current goals and constraints. The integration of planning and acting is critical. Models used for planning are descriptive (*know-what*) while models necessary for acting are operational (*know-how*). Ingrand and Ghallab classify robot architectures in four groups: (1) hierarchical, (2) reactive, (3) teleo-reactive, and (4) open. Hierarchical architectures are the most common choice. Components are organized into layers having different levels of abstraction. They identify as a challenge the implementation of deliberation functions that span multiple levels. Reactive architectures have limited or no need for deliberation. They are built based on input-output mappings implemented by automata. Teleo-reactive architecture integrates planning and acting along different levels of abstraction using a unified representation. A primary concern of these approaches is their scalability. Open architectures are suitable for complex open environments that require of knowledge processing capabilities. These architectures allow a robot to integrate models and knowledge from the web.

In Figure 1, a depiction of a hierarchical deliberative architecture based on Ingrand and Ghallab's proposal shows how the different deliberation functions are part of the decisional autonomy of the robot. A particular instantiation of this architecture implementing some of the deliberation functions applied to a industrial logistics scenario can be found in [51, 15]. In this architecture, there is more than one knowledge base enabling decision-making and reasoning tasks at the different levels of the abstraction hierarchy. Connectors are represented by buses rather than individual pairs to show the intent of integrative approaches.

In [12], Cox et al. present MIDCA cognitive architecture. MIDCA is based on *computational meta-*

cognition processes analogous to an *action-perception cycle*. The difference is that inputs are not interpretations of external stimuli but cognition states and instead of acting in the world, actions regulate cognitive activity. Cox et al. identify three fundamental forms: (1) explanatory, (2) immediate, and (3) anticipatory. Explanatory responds to *failures* in already executed cognitive processes, immediate is a form of runtime introspection, and anticipatory is self-directed foresight. The purpose of explanatory meta-cognition is to diagnose what caused a cognitive-level failure by formulating a meta-level goal to correct the situation. Anticipatory meta-cognition is a form of goal reasoning over *suspended goals*, those that the agent could not achieve because of insurmountable constraints. Meta-cognitive functions appear in many robotic solutions, for example, RA implemented a form of meta-planning.

2.5.4 Knowledge-Enabled

In [39], Reiter subscribes to the knowledge representation hypothesis (KRH) of Smith [43]: " Any mechanically embodied intelligent process will be comprised of structural ingredients that (a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and (b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behavior that manifests that knowledge. " This hypothesis has dominated the KRR research [28] for many years. To a certain degree (see MAR higher cognitive abilities in the previous section) knowledge is a horizontal concern across all cognitive abilities for robots, especially when high levels of cognitive ability are required. The remark here is that knowledge-enabled robots use explicit models, as compared to implicit knowledge embedded within code and data artifacts.

In [49], Tenorth and Beetz study KR for autonomous robots. They discuss *KnowRob* a *Knowledge Processing System* (KPS), advocating for the use of this paradigm to address the problem of filling the gap between information-rich low-level sub-symbolic representations and abstract high-level descriptions. To address this problem, their approach engineers common representational structures and appropriate inference procedures that enable posing and resolving queries necessary for the execution of abstract instructions by *interpreting* background knowledge and the context of the task at hand. An advantage of their approach is that it builds on top of already existing data and algorithmic robotic components by enriching them with semantic annotations. They acknowledge the advantages of logical approaches, regardless of whether they take the form of action theories, planning-oriented descriptions, or probabilistic representations, for modeling actions, preconditions, and effects that entail control, prediction, and analysis tasks. Unfortunately, most of the time they rise applicability concerns related to high or unpredictable computational costs. For real-world applications, the number of atoms in the axiomatizations grows rapidly forcing the modeler to keep a high level of abstraction but requiring the agent to resolve the fine level of detail required for the execution of actions in the environment. By preserving the already efficient sub-symbolic and ad hoc data structures and behaviors, there is no impact on efficiency nor information loss due to the abstraction required for symbolic inference. They refer to this constructs as computing *symbolic views* as needed or *on-demand abstractions*. Rather than attempting to aim for a complete and consistent world model the knowledge representation is a semantic *integration layer*. Some of the insights from their work are: (1) No fixed levels of abstraction, depending on the use case at hand the right number of abstraction levels differ; (2) a knowledge base should reuse data structures of the robot's control program; (3) symbolic knowledge bases are useful, but not sufficient, a *shallow* symbolic representation in the knowledge base interpret rich detailed information relevant for appropriate behavior; (4) robots need multiple inference methods, temporal reasoning of events can be combined with ontological reasoning and spatial reasoning, for example; (5) evaluating a knowledge base is difficult, for example, run-time of queries, or scalability concerns, a major challenge resides on

the quantification of these aspects. Key to the knowledge processing system is the use of the W3C web ontology language [31] (OWL) to represent a *common ontology* that is used as a *glue* to unify several reasoning algorithms.

Based on web services, other research initiatives have successfully developed platforms that allow robots to query, process, or share knowledge according to the task at hand [53, 6]. Open architectures exploit *ontology-based* approaches to augment the level of autonomy in robots. For a review on the field we refer the reader to [34], where Olivares-Alarcos et al. classify and compare different ontology-based approaches.

3 Towards a meta-architecture for autonomy

In this section, we connect the evidence presented in previous sections to build the case for a formal cognitive meta-architectural description. A meta-architecture is not an architecture but a description of a family of architectures. It should not be confused with meta cognition [11], although we consider meta cognition as an important architectural driver. A meta-architecture is closely related to the concept of architectural templates.

We summarize the analysis dimensions presented in the previous sections. LTRA in robots requires of high levels of *dependability*. The research within this dimension is highly mature, currently providing effective techniques and industrial standards. *MDE* can support the life cycle of robotic solutions by means of methodologies and automation of model transformations and code artifacts synthesis. Correct-by-construction can significantly reduce the cost of certification. Additionally, models facilitate reproducibility and widespread usage of research outputs by practitioners. Almost independently of the level of autonomy, *autonomic computing* provides robust and adaptive behavior. Explicit architectural modeling of autonomic functions is fully amenable to MDE and vice versa. Regarding the *software engineering* dimension, we consider that architectures for autonomy should aim to be expressed in ADLs with clear and sound semantic theories that facilitate analysis and runtime verification at the same time aim to reduce the semantic gap for the robotics field. Specific applications demanding a particular level of *autonomy* will benefit from a meta-architecture that is able to derive the best compromise between *cognitive abilities* and cost. The robust integration of the different cognitive functions remains the main challenge, and it is a significant threat towards dependable robotic solutions. Finally, besides enabling complex behavior and rational decision-making, knowledge-enable robotics contribute towards the trustworthiness of a robotic system operating in complex environments by facilitating its transparency and explainability.

Regarding cognition, our vision includes, although is not limited to, the possibility of an open community standard model of cognitive computer that can ease the transition to a model-fueled ecosystem. There is already important progress in different dimensions necessary to achieve such conceptual and technological platform. This means that such developments can be built on top of existing paradigms with arguably minimal effort. One advantage of having an open standard at a meta-model level would mean that there is no strict requirement to collapse the diversity of approaches into a common implementation, instead each particular platform would be free to implement the standard using its own conceptual frameworks and toolchains.

As far as *acting* is concerned, we assess that there are already widely and well-proven execution engines including those provided in RobMoSys, and PRS. In particular, RobMoSys is an excellent candidate as a basis for implementing the required cognitive views. It is a technologically sound platform from the point of view of software engineering. One important remark is that the aim here would be to

close the semantic gaps that limit the implementation of higher-level cognitive abilities. CRAM is also a visionary work that is already in the same path. The formal model execution engine HIPPO is addressing the semantic gaps between high-level models and code artifacts through automated code generation based on the semantic theory of Petri nets. One possible obstacle toward higher cognitive functions allowing complex decision-making and reasoning is that the research effort has so far concentrated on expressive languages directed to humans. A roboticist has to a great extent solve the *control logic* that then is encoded in an expressive language, whether OP procedures (OpenPRS), behavior trees (RobMoSys), or the formal specification language FIACRE (HIPPO). We should aim to transit, progressively, to action theories. The control logic should be fully produced by the artificial agent, not a human programmer.

The challenges of integrating multiple cognitive deliberation functions as identified by Ingrad and Ghallab [22] give ample opportunities for research. It is still an art to come out with effective ways to handle the integration of cognitive functions. The current abundance of cognitive architectures provides a rich landscape for drawing paradigms of cognition to hitherto mainly engineering-motivated endeavors. In the field of cognitive architectures, there is already a call for a standard model out of community consensus [26]. We consider that a standard model for robot autonomy with higher-level deliberation functions is feasible. If realized as a formal model using ADLs, it can be grounded by automatic means fully or to a great extent to the currently sound and efficient technologies. One possible approach is to consider the integration of cognitive functions in the context of meta-reasoning layers. An approach based on MIDCA architecture would allow modeling cognitive process as mental *acts*. The advantage of this unifying approach is that we already have much of the necessary machinery in place. There is no apparent barrier using action theories to model cognitive processes.

Standardized knowledge representation models are not required as long as the appropriate transformation can be perform automatically. Meta-models, and consistent conceptualizations are more important. Meta-models should be abstract enough to be independent of the robot platform. As in many cognitive architectures, the skill level can be the base line. Underlying levels can also be considered, but we grant more importance to the exploitation of existing software modules that are widely used by the robotics community. In this context, it is important to adequate integrate dependable solutions out of unreliable components. Models that include computational complexity descriptions can be exploited by meta-reasoning processes to optimize internal cognitive state and modulate the cognitive load of the robot.

4 Concluding Remarks

Meta-models have proven to be effective for enabling high-level cognitive abilities in autonomous robots. Typically, however their scope do not include the integration of high-level deliberation functions corresponding to an explicit formal cognitive model. The existing MDE methods and technologies are currently mature enough to support the transition towards a full model-enabled development and operation of robotic solutions (programming-free above the skill level). Although, we consider important to work towards a standard formal model agreed by the community. Such standard model can then be automatically transformed into concrete robotic solutions with the possibility of automatically verifying and certifying their dependability. Our future work will aim at modeling a meta-architecture for LTRA.

5 Acknowledgments

Funded by LEAD project “Dependable Internet of Things in Adverse Environments”, TU Graz.

6 Bibliography

References

[1] Rachid Alami, Raja Chatila, Sara Fleury, Malik Ghallab & Félix Ingrand (1998): *An architecture for autonomy*. The International Journal of Robotics Research 17(4), pp. 315–337, doi:10.1177/027836499801700402.

[2] University of Applied Sciences Ulm (2009): *SmartSoft: Component and Toolchain for Robotics*. `http://smart-robotics.sourceforge.net/`.

[3] Algirdas Avizienis, J-C Laprie, Brian Randell & Carl Landwehr (2004): *Basic concepts and taxonomy of dependable and secure computing*. IEEE transactions on dependable and secure computing 1(1), pp. 11–33, doi:10.1109/TDSC.2004.2.

[4] Franz Baader (2003): *Appendix: description logic terminology*. The Description logic handbook: Theory, implementation, and applications, pp. 485–495.

[5] Michael Beetz, Lorenz Mösenlechner & Moritz Tenorth (2010): *CRAM - A Cognitive Robot Abstract Machine for everyday manipulation in human environments*. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp. 1012–1017, doi:10.1109/IROS.2010.5650146.

[6] Michael Beetz, Moritz Tenorth & Jan Winkler (2015): OPEN-EASE. In: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp. 1983–1990, doi:10.1109/ICRA.2015.7139458.

[7] Peter Bishop & Robin Bloomfield (1998): *A methodology for safety case development*. In F Redmill & T Anderson, editors: Industrial Perspectives of Safety-critical Systems: Proceedings of the Sixth Safety-critical Systems Symposium, Birminham, Springer, pp. 1–9, doi:10.1007/978-1-4471-1534-2_14.

[8] Davide Brugali & Patrizia Scandurra (2009): *Component-based robotic engineering (Part I) [Tutorial]*. IEEE Robotics Automation Magazine 16(4), pp. 84–96, doi:10.1109/MRA.2009.934837.

[9] Davide Brugali & Azamat Shakhimardanov (2010): *Component-Based Robotic Engineering (Part II)*. IEEE Robotics Automation Magazine 17(1), pp. 100–112, doi:10.1109/MRA.2010.935798.

[10] Cristiano Castelfranchi & Rino Falcone (2003): *From automaticity to autonomy: the frontier of artificial agents*. In: Agent autonomy, Springer, pp. 103–136, doi:10.1007/978-1-4419-9198-0_6.

[11] Michael T. Cox (2005): *Metacognition in computation: A selected research review*. Artificial Intelligence 169(2), pp. 104–141, doi:10.1016/j.artint.2005.10.009. Special Review Issue.

[12] Michael T Cox, Zahiduddin Mohammad, Sravya Kondrakunta, Ventaksampath Raja Gogineni, Dustin Dannenhauer & Othalia Larue (2021): *Computational Metacognition*. In: Proceedings of the Ninth Annual conference on Advances in Cognitive Systems, pp. 1–20, doi:10.48550/arXiv.2201.12885.

[13] Ruth A David & Maj Gen Paul Nielsen (2016): *Defense science board summer study on autonomy*. Technical Report, Defense Science Board Washington United States. DTIC number: AD1017790. `https://dsb.cto.mil/reports.htm`.

[14] Edson de Araújo Silva, Eduardo Valentin, José Reginaldo Hughes Carvalho & Raimundo da Silva Barreto (2021): *A survey of Model Driven Engineering in robotics*. Journal of Computer Languages 62, p. 101021, doi:10.1016/j.cola.2020.101021. Available at `https://www.sciencedirect.com/science/article/pii/S2590118420300812`.

[15] Marco De Bortoli, Stalin Munoz Gutierrez & Gerald Steinbauer-Wagner (2021): *Diagnosis of hidden faults in the RCLL*. In: 32 International Workshop on Principle of Diagnosis, p. 1. `https://www.hsu-hh.de/imb/wp-content/uploads/sites/677/2021/09/DX-2021_paper_15.pdf`.

[16] Huáscar Espinoza, Marco Mahn & Anna Principato (2020): COMPOSABLE MODELS AND SOFTWARE FOR ROBOTICS SYSTEMS. Technical Report H2020–ICT–732410, RobMoSys – A European Union's Horizon 2020 funded project. `https://robmosys.eu/wp-content/uploads/2021/04/D6.8_Final.pdf`.

[17] Stan Franklin & Art Graesser (1997): *Is It an agent, or just a program?: A taxonomy for autonomous agents*. Berlin: Springer Verlag, doi:10.1007/BFb0013570.

[18] Alan G Ganek & Thomas A Corbi (2003): *The dawning of the autonomic computing era*. IBM systems Journal 42(1), pp. 5–18, doi:10.1147/sj.421.0005.

[19] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl & Peter Steenkiste (2004): *Rainbow: Architecture-based self-adaptation with reusable infrastructure*. Computer 37(10), pp. 46–54, doi:10.1109/ICAC.2004.1301377.

[20] Pierre-Emmanuel Hladik, Félix Ingrand, Silvano Dal Zilio & Reyyan Tekin (2021): HIPPO*: A Formal-Model Execution Engine to Control and Verify Critical Real-Time Systems*. Journal of Systems and Software 181, p. 111033, doi:10.1016/j.jss.2021.111033.

[21] Felix Ingrand (2014): *OPRS Development Environment Version 1.1b7*. Franois Flix Ingrand. `https://homepages.laas.fr/felix/publis-pdf/oprs.pdf`.

[22] Félix Ingrand & Malik Ghallab (2017): *Deliberation for autonomous robots: A survey*. Artificial Intelligence 247, pp. 10–44, doi:10.1016/j.artint.2014.11.003.

[23] Daniel Jackson, Martyn Thomas & Lynette I Millett (2007): *Software for dependable systems: Sufficient evidence?* National Academies Press, doi:10.17226/11923.

[24] Stuart Kent (2002): *Model driven engineering*. In: International conference on integrated formal methods, Springer, pp. 286–298, doi:10.1007/3-540-47884-1_16.

[25] Iuliia Kotseruba & John K Tsotsos (2020): *40 years of cognitive architectures: core cognitive abilities and practical applications*. Artificial Intelligence Review 53(1), pp. 17–94, doi:10.1007/s10462-018-9646-y.

[26] John E Laird, Christian Lebiere & Paul S Rosenbloom (2017): *A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics*. Ai Magazine 38(4), pp. 13–26, doi:10.1609/aimag.v38i4.2744.

[27] Jean-Claude Laprie (1985): *Dependable computing and fault-tolerance*. Digest of Papers FTCS-15 10(2), p. 124, doi:10.1109/FTCSH.1995.532603.

[28] Hector J Levesque & Ronald J Brachman (1985): *A fundamental tradeoff in knowledge representation and reasoning*. In Hector J Levesque & Ronald J Brachman, editors: Readings in Knowledge Representation, chapter 4, Morgan Kaufmann, pp. 41–70.

[29] Ivano Malavolta, Grace Lewis, Bradley Schmerl, Patricia Lago & David Garlan (2020): *How do you architect your robots? State of the practice and guidelines for ROS-based systems*. In: 2020 IEEE/ACM 42nd international conference on software engineering: software engineering in practice (ICSE-SEIP), IEEE, pp. 31–40, doi:10.1145/3377813.3381358. `https://ieeexplore.ieee.org/abstract/document/9276566`.

[30] Nenad Medvidovic & Richard N Taylor (2000): *A classification and comparison framework for software architecture description languages*. IEEE Transactions on software engineering 26(1), pp. 70–93, doi:10.1109/32.825767.

[31] Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg & Uli Sattler (2009): *OWL 2 web ontology language: Structural specification and functional-style syntax*. W3C recommendation 27(65), p. 159.

[32] Nicola Muscettola, P.Pandurang Nayak, Barney Pell & Brian C. Williams (1998): *Remote Agent: to boldly go where no AI system has gone before*. Artificial Intelligence 103(1), pp. 5–47, doi:10.1016/S0004-3702(98)00068-X. Available at `https://www.sciencedirect.com/science/article/pii/S000437029800068X`. Artificial Intelligence 40 years later.

[33] Pandurang Nayak, Douglas E Bernard, Gregory Dorais, Edward B Gamble Jr, Bob Kanefsky, James Kurien, William Millar, Nicola Muscettola, Kanna Rajan, Nicolas Rouquette, Benjamin D Smith, William Taylor & Yu wen Tung (1999): *Validating the DS-1 remote agent experiment*. In Michael Perry, editor: Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS '99), 440, Noordwijk, Netherlands : European Space Agency, pp. 349–356. Bibliographic code:1999ESASP.440..349N. `https://articles.adsabs.harvard.edu//full/1999ESASP.440..349N/0000349.000.html`.

[34] Alberto Olivares-Alarcos, Daniel Beßler, Alaa Khamis, Paulo Gonçalves, Maki K Habib, Julita Bermejo-Alonso, Marcos Barreto, Mohammed Diab, Jan Rosell, João Quintas, Joanna Olszewska, Hirenkumar Nakawala, Edison Pignaton, Amelie Gyrard, Stefano Borgo, Guillem Alenyà, Michael Beetz & Howard Li (2019): *A review and comparison of ontology-based approaches to robot autonomy*. *The Knowledge Engineering Review* 34, doi:10.1017/S0269888919000237.

[35] Peyman Oreizy, Michael M Gorlick, Richard N Taylor, Dennis Heimhigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S Rosenblum & Alexander L Wolf (1999): *An architecture-based approach to self-adaptive software*. *IEEE Intelligent Systems and Their Applications* 14(3), pp. 54–62, doi:10.1109/5254.769885.

[36] Zenon W Pylyshyn (1988): *Computing in cognitive science*. University of Western Ontario, Centre for Cognitive Science London. `http://www.cse.buffalo.edu/~rapaport/575/F01/Pylyshyn89.pdf`.

[37] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler & Andrew Y Ng (2009): *ROS: an open-source Robot Operating System*. In: *ICRA workshop on open source software*, 3, Kobe, Japan, p. 6. `http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf`.

[38] Ansgar Radermacher, Matteo Morelli, Mahmoud Hussein & Reda Nouacer (2021): *Designing Drone Systems with Papyrus for Robotics*. In: *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*, DroneSE and RAPIDO '21, Association for Computing Machinery, New York, NY, USA, pp. 29–35, doi:10.1145/3444950.3444956.

[39] Raymond Reiter (2001): *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT press, doi:10.7551/mitpress/4074.001.0001.

[40] John Rushby (2008): *Runtime certification*. In: *International Workshop on Runtime Verification*, Springer, pp. 21–35, doi:10.1007/978-3-540-89247-2_2.

[41] Douglas C Schmidt (2006): *Model-driven engineering*. *Computer-IEEE Computer Society-* 39(2), p. 25, doi:10.1109/MC.2006.58.

[42] Lawrence Shattuck (2015): *Transitioning to Autonomy: A Human Systems Integration Perspective*. `https://human-factors.arc.nasa.gov/workshop/autonomy/download/presentations/Shaddock%20.pdf`.

[43] Brian Smith (1982): *Reflection and Semantics in a Procedural Language*. Ph.D. thesis, Tech. Report MIT/LCS/TR-272. MIT, Cambridge, MA.

[44] SPARC (2020): *Robotics 2020 Multi-Annual Roadmap for Robotics in Europe*. Technical Report Horizon 2020 Call ICT-2017 (ICT-25, ICT-27 & ICT-28), SPARC, The Partnership for Robotics in Europe.

[45] Dennis Stampfer, Alex Lotz, Matthias Lutz & Christian Schlegel (2016): *The SmartMDSD toolchain: An integrated MDSD workflow and integrated development environment (ide) for robotics software*. *Journal of Software Engineering for Robotics (JOSER)* 7(1), pp. 3–19, doi:10.6092/JOSER_2016_07_01_p3.

[46] Andreas Steck, Alex Lotz & Christian Schlegel (2011): *Model-driven engineering and run-time model-usage in service robotics*. *ACM SIGPLAN Notices* 47(3), pp. 73–82, doi:10.1145/2189751.2047875.

[47] Gerald Steinbauer (2016): *Dependability of Autonomous Systems*. Ph.D. thesis, Faculty of Computer Science and Biomedical Engineering, Graz University of Technology. Cumulative Habilitation thesis.

[48] Gerald Steinbauer, Stefan Loigge & Clemens Mühlbacher (2016): *Supervision of Hardware, Software and Behavior of Autonomous Industrial Transport Robots*. In: *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 298–300, doi:10.1109/QRS-C.2016.44.

[49] Moritz Tenorth & Michael Beetz (2017): *Representations for robot knowledge in the KnowRob framework*. *Artificial Intelligence* 247, pp. 151–169, doi:10.1016/j.artint.2015.05.010. Available at `https://www.sciencedirect.com/science/article/pii/S0004370215000843`. Special Issue on AI and Robotics.

[50] Catherine Tessier (2017): *Robots autonomy: Some technical issues*. In W F Lawless, Ranjeev Mittu, Donald Sofge & Stephen Russell, editors: *Autonomy and Artificial Intelligence: A Threat or Savior?*, Springer, pp. 179–194, doi:10.1007/978-3-319-59719-5_8.

[51] Thomas Ulz, Jakob Ludwiger & Gerald Steinbauer (2019): *A robust and flexible system architecture for facing the RoboCup Logistics League challenge*. In: *Robot World Cup*, Springer, pp. 488–499, doi:10.1007/978-3-030-27544-0_40.

[52] Frank Van Harmelen, Vladimir Lifschitz & Bruce Porter (2008): *Handbook of knowledge representation*. Elsevier.

[53] Markus Waibel, Michael Beetz, Javier Civera, Raffaello D'Andrea, Jos Elfring, Dorian Gálvez-López, Kai Häussermann, Rob Janssen, J.M.M. Montiel, Alexander Perzylo, Björn Schießle, Moritz Tenorth, Oliver Zweigle & René Van De Molengraft (2011): *RoboEarth*. *IEEE Robotics Automation Magazine* 18(2), pp. 69–82, doi:10.1109/MRA.2011.941632.

[54] Yingxu Wang, George Baciu, Yiyu Yao, Witold Kinsner, Keith Chan, Bo Zhang, Stuart Hameroff, Ning Zhong, Chu-Ren Hunag, Ben Goertzel, Duoqian Miao, Kenji Sugawara, Guoyin Wang, Jane You, Du Zhang & Haibin Zhu (2010): *Perspectives on cognitive informatics and cognitive computing*. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)* 4(1), pp. 1–29, doi:10.4018/jcini.2010010101.

[55] Charles B Weinstock, John B Goodenough & John J Hudak (2004): *Dependability cases*. Technical Report, Carnegie-mellon University, doi:10.1184/R1/6572984.v1. CMU/SEI-2004-TN-016.

[56] Brian C Williams & P Pandurang Nayak (1996): *A model-based approach to reactive self-configuring systems*. In Dan Weld, Bill Clancey, Usama M. Fayyad & Howard Shrobe, editors: *Proceedings of the national conference on artificial intelligence (AAAI'96)*, 2, AAAI Press, Portland, Oregon, pp. 971–978, doi:10.5555/1864519.1864531.

[57] Du Zhang & Éric Grégoire (2016): *Learning through Overcoming Inconsistencies*. In: *2016 27th International Workshop on Database and Expert Systems Applications (DEXA)*, pp. 121–128, doi:10.1109/DEXA.2016.038.